Algorithms and optimized implementations in context of circular string and relevant web security.

Samir Uzzaman, Mohammad

Awarding institution:
King's College London

KINGS COLLEGE LONDON

PHD THESIS

Algorithms and optimized implementations in context of circular string and relevant web security

Author:
Mohammad SAMIR UZZAMAN

Supervisor:
Professor Costas ILLIOPOULOS

Examiner:
Dr Ida Pu

Examiner:
Dr Ahcène Bounceur

A report submitted in fulfilment of the requirements for PhD in the

Algorithms and BioInformatics

Department of Informatics

March 9, 2017

Declaration of Authorship

I declare that this thesis titled, "Algorithms and optimized implementations in context of circular string and relevant web security" and the work presented in it are my own, except where explicitly stated otherwise. I have published following articles as a co-author during my research. The materials and concepts from these publications are necessarily presented with this thesis.

- M. A. R. Azim, C. S. Iliopoulos, M. S. Rahman and M. Samiruzzaman*, "A Simple, Fast, Filter-Based Algorithm for Approximate Circular Pattern Matching", in IEEE Transactions on NanoBioscience, vol. 15, no. 2, pp. 93-100, March 2016.

- Costas S. Iliopoulos, M. Samiruzzaman, M Sohel Rahman, Steven Watts Client side web based application for search space reduction in approximate circular pattern matching. In Bioinformatics Research and Applications - 12th International Symposium URL:http://alan.cs.gsu.edu/isbra16/ sites/default/files/short/ ISBRA_2016_paper_98.pdf

- Moudhi Aljamea, Ljiljana Brankovic, Jia Gao, Costas S. Iliopoulos, and M. Samiruzzaman. Smart meter data analysis. ICC '16 Proceedings of the International Conference on Internet of things and Cloud Computing. ACM New York, NY, USA ©2016 SBN: 978-1-4503-4063-2 doi:10.1145/2896387.2896407 Article No. 22

- Moudhi Aljamea, Costas S. Iliopoulos, and M. Samiruzzaman. Detection Of URL In Image Steganography. ICC '16 Proceedings of the International Conference on Internet of things and Cloud Computing. ACM New York, NY, USA ©2016 ISBN: 978-1-4503-4063-2 doi:10.1145/2896387.2896408 Article No. 23

- Moudhi AlJamea, Costas S. Iliopoulos, M. Samiruzzaman Effective Solutions For Most Common Vulnerabilities In Web Applications, 2016 SAI Intelligent Systems Conference INTELLISYS2016. Proceedings available at https://www.scribd.com/document/326609639/IntelliSys2016Proceedings

- Costas S. Iliopoulos, Mai Alzamel, Mudhi Aljamea, M Samiruzzaman Fast Fingerprint Identification Approach Via Circular String Matching Filters - Accepted on international conference - Future Technologies Conference 2016

- Md. Aashikur Rahman Azim, Costas S. Iliopoulos, M. S. Rahman, and M . Samiruzzaman. A filter-based approach for approximate circular pattern matching. In Bioinformatics Research and Applications - 11th International Symposium, pages 24-35, 2015.

- Md. Aashikur Rahman Azim, Costas S. Iliopoulos, M. Sohel Rahman and M. Samiruzzaman SimpLiFiCPM: A Simple and Lightweight Filter-Based Algorithm for Circular Pattern Matching International Journal of Genomics Volume 2015 (2015), Article ID 259320 http://dx.doi.org/10.1155/2015/259320

- Md. Aashikur Rahman Azim, Costas S. Iliopoulos, M. S. Rahman, and M . Samiruzzaman. A fast and lightweight filter-based algorithm for circular pattern matching. In Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics, BCB 14 California USA 2014, pages 621-622, 2014.

The above are the co-joint works necessarily presented in the thesis. The candidate contributed on average at least 75 percent on each paper mentioned above. The rest of the material of this thesis is entirely candidate's contribution.

KINGS COLLEGE LONDON

Abstract

Faculty of Natural And Mathematical Sciences

Department of Informatics

PhD Thesis

Algorithms and optimized implementations in context of circular string and relevant web security

by Mohammad SAMIR UZZAMAN

This thesis deals with the efficient algorithms and implementations on circular pattern matching along with the web tools and relevant security in practical perspective. The contribution has been attributed by developing efficient algorithms and implementing web based tool as an application on circular pattern matching. Extensive experiments on circular pattern matching has been conducted in traditional desktop based programming environment. The experimental results have been found to be excellent. An web based software tool on circular pattern matching has also been implemented. While developing the web based tool, the research progressed to the area of web security and vulnerability in some extent. The thesis addressed that web based security issues by providing the solutions in both algorithmic and practical perspective.

An introduction of this thesis has been presented in Chapter 1. This chapter gives an overview and brief literature review of the problems those have been addressed in the thesis. The algorithms and implementations on exact circular string matching has been presented in Chapter 2. Approximate circular string matching is a practical variation of circular pattern matching. The algorithms, implementations and experimental results on approximate circular string matching has been presented in Chapter 3. Circular string matching research on biological perspective is very new, so there is a lack of web based efficient tool for the biologists on circular string matching. The web based tool has been developed and presented in Chapter 4. Web based tool attracts inherent Internet security and vulnerability. The known vulnerability issue in development perspective has been presented in Chapter 5. Circular pattern matching algorithms have been applied to solve One to Many fingerprint problems in Chapter 6 by adapting the algorithms in Chapter 2 and Chapter 3. The further extension in the area of web security has been done in Chapter 7 which deals with the URL in Steganography.

Acknowledgements

I would like to thank Professor Costas Illiopoulos, my supervisor, for his invaluable guidance and support throughout my studies. I would also like to express my gratitude to my co-supervisor Dr. Solon Pissis.

I would like to thank all of my co-authors of the papers for their wonderful cooperation.

I would like to thank Professor Sohel Rahman, Dr. Pradipta Mitra, Dr. Carl Burton, Dr. Manal Mohamed and Fatima Vayani for their constructive feedback on my thesis.

I have done some collaborative works with few researchers who were funded by Inspire Project run by British Council. I would like to thank British Council and Inspire Project for their contribution.

Finally, I would like to thank my family, my wife Zinnatun Nahar Papsi, my daughter Rodela Prokriti Samir and my son Rishan Abhirup Samir. Without their sacrifice of endless evenings and weekends, it would not have been possible for me to pursue my PhD.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides a brief introduction to the content of this thesis. I have developed efficient algorithms and implemented web based tools on circular pattern matching. While developing these tools, my research progressed to the area of web security and vulnerability in some extent. I have addressed those security issues by providing the solutions from both algorithmic and practical perspectives. I presented my algorithms and implementations on Exact Circular Pattern matching in Chapter 2 and Approximate circular pattern matching in Chapter 3. Circular string matching research on biological perspective is very new, so there is a lack of efficient web based tools for biologists to solve circular string matching problem. I have developed and presented my web based tool on Chapter 4. Web based tools introduce inherent security and vulnerability issues. I addressed the known vulnerability issues from a development perspective in Chapter 5. I have applied circular pattern matching algorithms and solved One to Many fingerprint problems in Chapter 6. I have further extended my work in the area of web security in Chapter 7

1.1 Circular pattern matching

1.1.1 Overview

The circular pattern, denoted by $\mathcal{C}(\mathcal{P})$, corresponding to a given pattern $\mathcal{P} = \mathcal{P}_1 \ldots \mathcal{P}_m$, is formed by connecting $\mathcal{P}_1$ with $\mathcal{P}_m$ and forming a cycle. This gives us the notion where the same circular pattern can be seen as m different linear patterns, which would all be considered equivalent. In the Circular Pattern Matching (CPM) problem, one is interested in pattern matching between the text $\mathcal{T}$ and the circular pattern $\mathcal{C}(\mathcal{P})$ of a given pattern $\mathcal{P}$. This views $\mathcal{C}(\mathcal{P})$ as a set of m patterns starting at positions $j \in [1 : m]$ and wrapping around the end. In other words, in CPM, I search for all rotations of a given pattern in a given text. While CPM itself is a variant of the classic pattern matching problem, further useful variations are possible. The requirement of approximation or allowing errors in CPM is also important and is natural because of the possibility of error in data. Instead of locating exact occurrences of all the rotations, approximation allows a threshold of errors within each occurrence. This natural extension gives us the approximate circular pattern matching (ACPM) problem which is more interesting and challenging.

In Chapter 2 I present a fast algorithm for the exact circular pattern matching problem based on some filtering techniques. In Chapter 3 I present a fast algorithm for the approximate circular pattern matching. In particular, I employ a number of simple and effective filters to preprocess the given pattern and the text. After this preprocessing, I get a text of reduced length on which I can apply existing state of the art algorithms to get the occurrences. I have done experimental studies to compare our algorithm with the state of the art algorithms and the results are found to be excellent. The algorithm turns out to be much faster in practice because of the huge reduction in the search space through filtering. Also, the filtering technique is simple and lightweight.

1.1.2 Brief literature review

Perhaps the first attempt to solve the problem of circular pattern matching has been documented in [6], where an $\mathcal{O}(n)$-time algorithm is presented. A naive solution with quadratic complexity would be to apply a classical algorithm for searching a finite set of strings on the *trie* of rotations of $\mathcal{P}$ after constructing it. The approach presented in [6] preprocesses $\mathcal{P}$ by constructing a suffix automaton of the string $\mathcal{PP}$, because, every rotation of $\mathcal{P}$ is a factor of $\mathcal{PP}$. Then, by feeding $\mathcal{T}$ into the automaton, the lengths of the longest factors of $\mathcal{PP}$ occurring in $\mathcal{T}$ can be found by the links followed in the automaton in time $\mathcal{O}(n)$. In [7], an optimal average-case algorithm for CPM has been presented. In particular, here the authors have shown that the average-case lower bound for the (linear) pattern matching of $\mathcal{O}(n \log_\sigma m/m)$ also holds for CPM, where $\sigma = |\Sigma|$. Recently, Chen et al. [8] have exploited word-level parallelism to present two fast average-case algorithms. Very recently, I and my coauthors have presented a filter-based approach to solve the problem in [9] and [10]. Our approach in [10, 9] turns out to be highly effective in practice. In fact, as it will be clear shortly, this thesis extends our approach of [10, 9] to solve the approximate version of the problem. Notably, a preliminary version of our work has been presented at [11]. The further works has been published on the journal [12].

To the best of my knowledge, the approximate version of the problem has not received much attention in the literature until it has been studied in [4] very recently. In [4], Barton et al. have first presented an efficient algorithm for CPM, i.e., for the exact version of the problem which runs in $\mathcal{O}(n)$ time on average. Based on the above, they have subsequently devised fast average-case algorithms (ACSMF-Simple) for ACPM, i.e., approximate circular string matching allowing k-mismatches. They have built a library for ACSMF-Simple algorithm that is freely available at [13].

Indexing circular patterns [14] and variations of circular string matching under the

edit distance model [15] have also been considered in the literature. Apart from being interesting from a purely combinatorial point view, CPM has applications in areas such as geometry, astronomy and computational biology. Circular patterns occur in the DNA of viruses [16, 17], bacteria [18], eukaryotic cells [19], and archaea [20]. As a result, as has been noted in [21], algorithms on circular strings seem to be important in the analysis of organisms with such structure. Circular strings have previously been studied in the context of sequence alignment. In [22], basic algorithms for pairwise and multiple circular sequence alignment have been presented. These results have later been improved in [23], where an additional preprocessing stage is added to speed up the execution time of the algorithm. In [24], the authors also have presented efficient algorithms for finding the optimal alignment and consensus sequence of circular sequences under the Hamming distance metric. For further details on the motivation and applications of this problem in computational biology and other areas the readers are referred to [16, 17, 18, 19, 20, 21, 22, 23, 24] and references therein.

1.1.3 Preliminaries

Let Σ be a finite *alphabet*. An element of Σ^* is called a *string*. The length of a string w is denoted by $|w|$. The empty string ϵ is a string of length 0, that is, $|\epsilon| = 0$. Let $\Sigma^+ = \Sigma^* - \{\epsilon\}$. For a string $w = xyz$, where x, y and z are called a *prefix*, *factor* (or equivalently, *substring*), and *suffix* of w, respectively. The i-th character of a string w is denoted by $w[i]$ for $1 \leq i \leq |w|$, and the factor of a string w that begins at position i and ends at position j is denoted by $w[i:j]$ for $1 \leq i \leq j \leq |w|$. For convenience, let us assume $w[i:j] = \epsilon$ if $j < i$. Let k-factor is a factor of length k.

A circular string of length m can be viewed as a traditional linear string which has the left-most and right-most symbols wrapped around and stuck together. Under this notion, the same circular string can be seen as m different linear strings, which would all be considered equivalent. Given a string $\mathcal{P}$ of length m, denoted by $\mathcal{P}^i = \mathcal{P}[i : m]\mathcal{P}[1 : i-1]$, $0 < i < m$, the i-th *rotation* of $\mathcal{P}$ and $\mathcal{P}^0 = \mathcal{P}$.

Example 1 *Suppose I have a pattern $\mathcal{P} = atcgatg$. The pattern $\mathcal{P}$ has the following rotations (i.e., conjugates): $\mathcal{P}^1 = tcgatga, \mathcal{P}^2 = cgatgat, \mathcal{P}^3 = gatgatc, \mathcal{P}^4 = atgatcg, \mathcal{P}^5 = tgatcga, \mathcal{P}^6 = gatcgat$.*

The *Hamming distance* between strings $\mathcal{P}$ and $\mathcal{T}$, both of length n, is the number of positions i, $0 \leq i < n$, such that $\mathcal{P}[i] \neq \mathcal{T}[i]$. Given a non-negative integer k, let us write $\mathcal{P} \equiv_k \mathcal{T}$ or equivalently say that $\mathcal{P}$ k-matches $\mathcal{T}$, if the Hamming distance between $\mathcal{P}$ and $\mathcal{T}$ is at most k. In biology, the *Hamming distance* is popularly referred to as the *Mutation distance*. A little mutation could be considered and in fact anticipated while finding the occurrences of a particular (circular) virus in a carrier's DNA sequence. This scenario in fact refers to *approximate circular pattern matching* (ACPM). If, $k = 0$, then I get the exact CPM, i.e., mutations are not considered. Note that in this setting, ACPM also returns all the occurrences returned by CPM; it computes the occurrences allowing **up to** k mismatches/mutations.

I consider the DNA alphabet, i.e., $\Sigma = \{a, c, g, t\}$. In my approach, each character of the alphabet is associated to a numeric value as follows. Each character is assigned a unique numbers from the range $[1...|\Sigma|]$. Although this is not essential, I conveniently assign the numbers from the range $[1...|\Sigma|]$ to the characters of Σ following their inherent lexicographical order. Let us use $num(x), x \in \Sigma$ to denote the numeric value of the character x. So, I have $num(a) = 1, num(c) = 2, num(g) = 3$ and $num(t) = 4$,. For a string S, I use the notation S_N to denote the numeric representation of the string S; and $S_N[i]$ denotes the numeric value of the character $S[i]$. So, if $S[i] = g$ then $S_N[i] = num(g) = 3$. The concept of circular string and their rotations also apply naturally on their numeric representations.

Example 2 *Suppose I have a pattern $\mathcal{P} = atcgatg$. The numeric representation of $\mathcal{P}$ is $\mathcal{P}_N = 1423143$. And this numeric representation has the following rotations: $\mathcal{P}_N^1 = 4231431$, $\mathcal{P}_N^2 = 2314314$, $\mathcal{P}_N^3 = 3143142$, $\mathcal{P}_N^4 = 1431423$, $\mathcal{P}_N^5 = 4314231$, $\mathcal{P}_N^6 = 3142314$.*

1.2 Circular pattern matching for fingerprint identification

1.2.1 Overview

Fingerprints are used in many fields for several purposes. A typical fingerprints systems database may consists of millions of fingerprints template. Accordingly, improving matching performance with large database has been taken under consideration in the recent research. In this work, I proposed an efficient database indexing technique using 4 fast and lightweight filters. The approach reduces the number of nominated fingerprints before matching stage by applying the 4 filters on the database gradually. The introduced algorithm is robust and fast which is done in linear time regarding the fingerprints rotation.

Automated Fingerprint Recognition Systems (AFRS) usually deployed in two modes (verification and identification). Furthermore, these systems can be found in many applications such as managing the employees attendance, border control in airports, access control to sensitive data and so on. The verification recognition mode is a one to one matching procedure that matches the scanned fingerprint to the same registered fingerprint in the database [25]. On the other hand, identification recognition mode is a one to many matching procedure that identify only a number of fingerprints from which are saved in the database and matches the scanned fingerprint [25]. In Identification recognition mode, the database might contain a large volume of data due to the system requirements. For instance, the immigration systems at the airports deal with millions of saved fingerprints. The most popular technique is indexing the fingerprint database to adopt a subset of nominated fingerprints.

There are two main classes in fingerprints indexing: the first class uses features called ridge orientation map and ridge frequency map [26], [27] whereas, the second class uses feature called minutiae. Generally, the first class is faster than the second one since it stores the feature in compact vectors. On the other hand, the second class is more accurate since the minutiae consists more distinguishable information of the

fingerprints than ridge orientation and frequency maps.

Combining both indexing approaches have been used to improve the performance regarding speed and accuracy at [28], [29]. Keeping in mind, the matching process speed can be affected by the accuracy level. Achieving higher accuracy may result in slowing the matching process [25], [30].

1.2.2 Brief literature review

Human fingerprints can be represented as composite of ridges and grooves on the fingertip skin. The human fingertip skin consists of ridges and furrows, these two combination form a sophisticated pattern by running the ridges and furrows in parallel lines and curves. The paper [31] mentioned three main fingerprint patterns called whorl, loop and arch. Henry's fingerprint classification system classified the fingerprints into eight classes: Plain Arch, Tented Arch, Left Slant Loop, Right Slant Loop, Plain Whorl, Double-Loop Whorl, Central-Pocket Whorl, and Accidental Whorl [32], [33].

Every fingerprint is highly immutable and unique, this uniqueness is defined by minutiae which can be explained as the combinations of local features such as ridge endings and ridge bifurcations, global features such as ridges and valleys. [34].

In fact, there is an old tool for identifying people, adopting the so-called ink-technique [25]. This technique defined through dabbing the fingers with ink to print the fingerprints on paper which later scanned to be a digital fingerprints. This is called off-line fingerprints approach that matches fingerprints by utilising the archived digital images. Despite the importance of the above technique in forensics field, it is not applicable for biometrics applications [35] which need real time solutions.

When it comes to solving the fingerprints identification problem in large databases, fingerprint indexing and fingerprint classification have been the most reliable solutions until now [36]. Henry classifications system [37] classified the fingerprints into five categories: left loop, right loop, whorl, arch, and tented arch. Nevertheless, these

number of categories are relativity small comparing to the number of the fingerprints in the databases. To explain, fingerprints indexing is to represent the fingerprints with feature vectors. This approach is deployed during the matching stage where a query fingerprint will be matched with the fingerprints in database which their vectors are similar to query vector [36]. Various algorithms have been proposed to improve fingerprints indexing which involve either global feature or minutiae feature.

1.2.3 Minutia feature

The main concept is to construct a robust and discriminant minutia with rotation and translation problems. In [38] and [39] the proposed algorithms are based on features of triangle or quadrangle geometry and selected simple hashing methods. However, these geometric characteristic has more sensitivity to distortion and noise. Whereas, the author of [40] designed a descriptor used a minutiae triplet. The features of triplet consists of the angles, the length and the ridge to be calculated between each minutiae pair. A false rate might be happening due to that the minutia is superior in discrimination. A pairwise enrolment is used to reduce the false rate between each input fingerprint and the stored fingerprints in the database using clustering transformation parameter which costs a lot of time.

Later, [38] developed the algorithm in [40] to increase the matching accuracy and the speed. They used a new minutiae triplets feature and effective ad-hoc geometric rules on the theory of triplets matching instead of pairwise enrollment. Therefore, an logarithm has been proposed in [41] to use a binary Minutia Cylinder Code (MCC) to be a minutia descriptor. This approach is proposed to encode the direction and location of neighbour minutiae around per minutia into a bit vector with fixed length. Although, MCC has higher accuracy and speed but it requires a higher dimension. Alternatively, Locality Sensitive Hashing (LSH) is utilised to search for MCC hypothesis correspondences rather than original quantisation strategy.However,it is not efficient for applications need high accuracy.

1.2.4 Global feature

The main idea of global feature is ridges information represented by specific points or orientation domain. In [42] and [43] the authors presented to find out selected points as features to be used in fingerprints indexing. There are draw backs of this method which needs fingerprints pre-alignment and there is a challenge to extract certain points from specified location from distorted image. In [44] and [26], the authors illustrated fingerprint indexing derived from orientation dormant to represents fingerprint feature vector. The main problem of this method is that the features are global and they are not able to distinguish the minutiae.

1.2.5 Circular string in fingerprint identification

I am interested in [45] that uses circular pattern matching. I explored the opportunity of solving one to many fingerprint identification problem by using some lightweight filters.

My proposed approach and contribution is to save the filter signature of each fingerprint in a database by using a circular string. The details are presented in chapter 6.

1.3 Web tool security

1.3.1 Overview

Most sites in current web spectrum are vulnerable. In fact, 86% of websites contain at least one 'serious' vulnerability according to the WhiteHat [46] Report 2015. Web developers worldwide are trying to provide solutions for user's functionality requirements. In most cases the security issues are not dealt with correctly while developing the product.

The web security is considered to be an expert area while in true sense the developers

needs to be aware of this issue during developing the applications. The functionality of an web application that fulfils the business requirements may have serious level of vulnerabilities. As a result the web applications becomes vulnerable to different kinds of attacks.

1.3.2 Brief literature review

I have developed both desktop and web based tool for circular patter matching in genome perspective. Web tool attracts inherent vulnerability of web. The common web vulnerabilities to date from software development perspective has been illustrated by the CWE/SANS top 25 most dangerous software errors report [47] in which web application vulnerabilities ranked the top. In addition, the industry scanning tool[48] gives the insight about the issues. According to the Open Web Application Security Project (OWASP) the most common web vulnerabilities are(SQL injection, Cross-site Scripting (XSS), Cross-Site Request Forgery (CSRF), Security Misconfiguration, Unvalidated Redirects and Forwards, No Account Lockout, Insecure Password Policy and HTTPS not enforced) as presented in their Top Ten Most Critical Web Application Security Risks Report in 2013 [49]. In most cases the hackers use these common vulnerabilities to get access to the system which later turns out to be disastrous for the organizations [47].

In Chapter 7, I presented the guidelines and solutions based on the understanding of vulnerabilities found in web tool.

1.4 Web tool for URL Steganography

1.4.1 Overview

Steganography is the science of hiding data within data. Steganography plays role in either for the good purpose of secret communication or with the intention of leaking

sensitive and confidential data or embedding malicious code or URL. Many different carrier file formats can be used to hide these data (network, audio, image etc). The most common Steganography carrier is embedding secret data within images as it is considered to be the easiest way to hide all types of files (secret files) within an image using different formats (another image, text, video, virus ,URL..etc). To the human eye, the changes in the image appearance with the hidden data can be imperceptible. In fact, images can be more than what we see with our eyes. Many solutions were proposed to help in detecting these hidden data but each solution has own strength and weakness either by the limitation of resolving one type of image along with specific hiding technique or in extracting the hidden data. My work intends to propose a novel detection approach that will concentrate on detecting any kind of hidden URL in the loss-less images and extract the hidden URL from the carrier image using the LSB (least significant bit) hiding technique.

1.4.2 Brief Literature Review

The word steganography is derived from the Greek words stegos meaning cover and grafia meaning writing [50]. Steganography and cryptography considered to be complementing each other rather than replacing each other. Cryptography is the art of scrambling messages to make it difficult to understand, whereas, Steganography is the art of hiding it to make it difficult to find. Therefore, steganography is an extra layer to support transferring secret information in a secure way. When steganography fails and the message can be detected, it is still of no use as it is encrypted using cryptography techniques [51].

Moreover, Steganographic techniques started ages ago back to ancient Greece. Starting by writing text on wax-covered tablets to shaving the head of a messenger and tattoo a message or image on the messenger's head, then, the hair will grew back, and the message will be undetected until the head was shaved again [52].

Since then, the science of steganography has developed significantly to more sophisticated techniques far more than their ancient predecessors, allowing a user to hide large amounts of information within image, audio files and even networks. In fact, in reality the main difference between the modern steganographic techniques and the previous one is only the form of carrier for the secret information. For instance, instead of using human skin and wooden tables, now it is being used by media files such as images,audio, video etc.

Although, within the daily discovery of a message hidden with in existing application, the new steganographic applications are being devised. The old methods are given new twists[52]. Therefore, there are many types of steganographic methods to hide secret data with new carriers, new hiding techniques or new type of secret data.

1.4.3 The concept of steganography

The concept of steganography is to embed data, which is to be hidden. This process requires three files:

First one is the secret message which is the information to be hidden. As mentioned before with the new steganography techniques almost any kind of data can be hidden. Second, is the cover file (carrier) that will hold the hidden information and as well almost any kind of files can be used as a carrier. Finally, the key file to find the hidden message and extract it from the cover file, the result of these three files is a file called (Stego file) as shown in Figure 1.1.

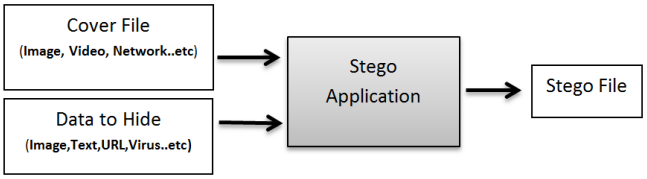


FIGURE 1.1: Stego application Scenario

The most common steganography technique is embedding messages within images as its considered to be the best carrier (Cover File) to hide all types of files within it. For example, hiding (another image, virus, URL, text, exe file , audio..etc) without changing its visible properties [53].

1.4.4 Steganography applications

Steganography can be used for good intentions in many useful ways for example to help in transferring secret data, copy rights control of materials and smart IDs where individuals' details are embedded in the photograph [54]. It can also be used in printed images where the data will be embedded before printing and the user can scan the printed image with a smart device and the embedded information will appear on the device, that can be useful in so many fields especially exhibitions and as a marketing tool to display the products information.

Nevertheless, hospitals are using Steganography to keep their patient's confidential data such as DNA sequences in a secret safe place where the access to it, is highly restricted.

Cyber-crime is believed to get benefited from transferring illegal data or embedding viruses and malicious URLs in carries and other harmful actions. Therefore, due to the rapid development of steganography methods and techniques, the steganography research area has gained the attention during the last decade.

Besides that, nowadays any person without any technical background can do harmful cyber crime actions with the support of the cyber crime ready made sophisticated softwares which are available online to everyone with no cost and easy to use [55]

1.4.5 Image steganography

Images can be more than what we see with our eyes. Using an image as a cover file is considered to be one of the most useful and cost effective technique [56], all the image steganographic techniques to hide data based on the structure of the most commonly used images format on the internet (GIF-graphics interchange format), (PNG-portable network groups) and (BMP- Bit Map Picture).

- ***Cover Image***: In steganography the original image that was chosen as a carrier for the secret data is called a cover image.
- ***Stego Image***: Is the result image of choosing the right cover image and embedding the secret data inside it.
- ***Stego Key***: The sender should have an algorithm for creating the stego image to embed the data, and the receiver should have the matching algorithm to extract the hidden data from that particular stego image and its called stegokey.

Image Embedding Process : Let C be the chosen *Cover Image*, and C' is the *Stego Image*, the *Stego Key* will be denoted as K, and the hidden message as M then:
$C \oplus M \oplus K \rightarrow C'$
as shown in Figure 1.2.

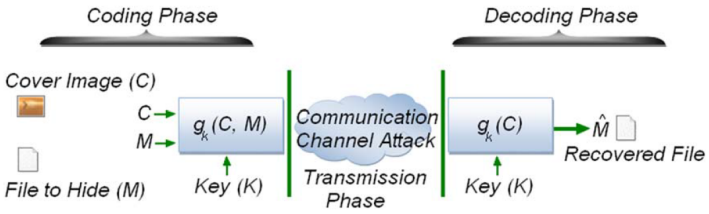


FIGURE 1.2: Image steganography embedding process [54]

The main challenge in image Steganography is that many image manipulation techniques might destroy the hidden message on any image because it might change

the feature of the stego-image. It might as well change the feature of the hidden message inside it. For example, cropping might crop the hidden message if it is located in one section of the image or corrupt it, rotation might give the receiver difficulty in finding the hidden message, filtering might destroy the hidden message completely and so on.

1.4.6 Current image steganography techniques

The stego file is generated by embedding the actual image file with the hidden data. The typical diagram is given here.

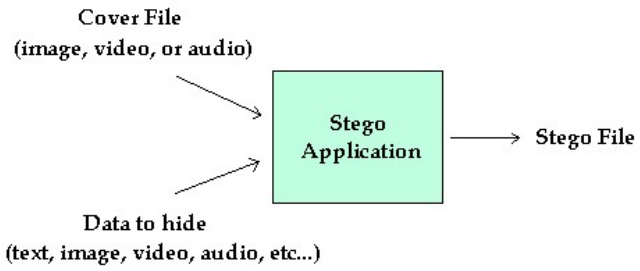


FIGURE 1.3: stegocode

Steganography embedding techniques can be divided into two groups. These are Spatial Domain and Frequency Domain. Spatial Domain embeds directly the secret data in the intensity of the image pixels usually with the least significant bit (LSB) in the image. Frequency Domain transforms the images and then the secret data is embedded in the transformed image [57].

According to the Review on current Methods and application of Digital Image Steganography 2015, it appears that the Spatial Domain is mostly used by researches in comparison to frequency domain as shown in Figure 1.4. Yet, there is a lot of scope

and opportunities to steganalysis and develop more effective methods for steganography [56].

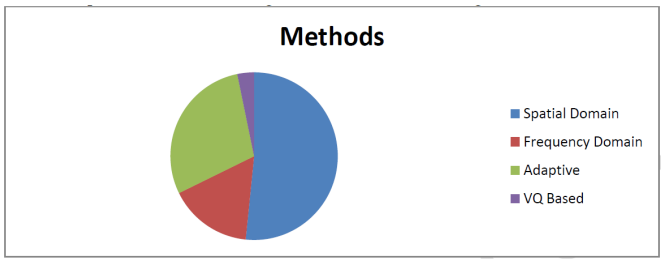


FIGURE 1.4: Stego methods

The focus of my work is on spatial domain. In spatial domain, the steganographer modifies the secret data and the cover image which involves re-encoding the least significance bits (LSBs) in the carrier image. To the human eye these changes in the image value of the LSB are imperceptible [58].

1.4.7 Least significant bits

It is the steganography approach of embedding data at the least significant bit (LSB) in the cover image.

Least Significant Bits (LSB) is considered to be one of the simplest approaches of embedding data in a cover image. This technique embeds the bits of the secret data directly into the least significant bit plane of the cover image [54].

On average, a small number of bits in an image will need to be modified to hide a secret data using the maximal cover size. In fact, the result of these changes are too small to be recognized by the human visual system (HVS). So the message is effectively hidden [50].

1.4.8 Steganalysis

Steganalysis is the main step in the steganography detecting technique to discover the hidden messages. It's the way of identifying the suspected medium , determine whether or not they have an embedded data into it, and, if possible, recovering that data. in other words 'Steganalysis is the science of attacking steganography in a battle that never ends' [54].

Stegnalysis can be done in different forms:

1. **The Steganography attack is known to the stegnalysis**: The cover file, the hidden message and the Steganography tool (algorithm) are all known to the steganlysis, then it should not be difficult to identify and locate the hidden message.

2. **Only the original file and the Cover file are known to stegnalysis**: The mission will be to compare the two files and identify the hidden message according to the pattern differences that are detected between the two files.

3. **Only the secret message only is known to the stegnalysis**: The mission will be looking for a known pattern in all the files and that may be very difficult to achieve.

4. **Only the cover file is known to the stegnalysis**: Similarly to the previous point it will be challenging to identify the hidden message location since it may be scattered to more than one place.

The image processing is usually the main technique used in building steganalysis programs to study different image manipulations such as translation , filtering, cropping and rotation. This is done by examining the cover image structure for first order statistics (histograms) or second order statistics (correlations between pixels, distance,

direction). The double compression of JPEG and the distribution of DCT (discrete cosine transform) coefficients can give hints on the use of DCT-based image steganography [54].

1.4.9 URL in image steganography

Embedding data in images is not a new technique. However, improving this method in image steganography is getting better and more sophisticated. One of the recent improvements is embedding a URL (Uniform Resource Locator) in the image least significant bits. The main reason behind embedding URL in an image instead of the whole secret data is that the URL occupies much lesser space in the carrier[59] and that helps hiding the data.

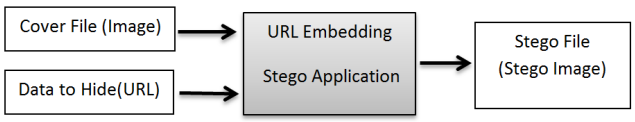


FIGURE 1.5: Stego process

Dell SecureWorks Counter Threat Unit^{TM} published a full analysis in June 2015 where they explained that malware authors are evolving their techniques to evade network and host-based detection mechanisms. stegoloader could represent an emerging trend in malware by the use of digital image steganography to hide malicious code.[60]. In previous report [61] Dell warned about Lurk Downloader where the malware was embed as URLs into an image file by inconspicuously manipulating individual pixels. I have presented my URL hiding and detection algorithms along with an web tool in Chapter 7.

Chapter 2

Exact Circular Pattern Matching

2.1 Introduction

Most recently in [4], the authors have presented an average-case algorithm for ECPM, with an average-case runtime of $\mathcal{O}(n)$ time, where n is the length of text $\mathcal{T}$.

In this chapter I present a fast algorithm for the exact circular pattern matching problem based on filtering techniques. In particular, I employ a number of simple and effective filters to preprocess the given pattern and the text. After this preprocessing, I get a text of reduced length on which any existing state of the art algorithm can be applied to get the occurrences. I have done experimental studies to compare my algorithm with the state of the art algorithms and the results are found to be excellent. My algorithm turns out to be much faster in practice because of the huge reduction in the search space through filtering. I present an algorithm with better experimental run time for searching patterns in filtered text string using the suboptimal average-case algorithm presented in [4] which runs in $O(n)$.

The rest of the chapter is organized as follows. Section 2.2 gives a preliminary description and problem definition along with the terminologies and concepts related to stringology that will be used in this thesis. In Section 2.3 I describe the filtering algorithms. Section 2.4 presents the experimental results.

2.2 Problem definition

Let $\sum$ be a finite *alphabet*. An element of $\sum^*$ is called a *string*. The length of a string w is denoted by $|w|$. The empty string ϵ is a string of length 0, that is, $|\epsilon| = 0$. Let $\sum^+ = \sum^* - \{\epsilon\}$. For a string $w = xyz$; x, y and z are called a *prefix*, *substring*, and *suffix* of w, respectively. The i-th character of a string w is denoted by $w[i]$ for $1 \leq i \leq |w|$, and the substring of a string w that begins at position i and ends at position j is denoted by $w[i:j]$ for $1 \leq i \leq j \leq |w|$. For convenience, let $w[i:j] = \epsilon$ if $j < i$. A *string* y is called a *factor* of w if $w = xyz$ for $x, z \in \sum^*$; in this case, the string y occurs at position $|x| + 1$ in w. The factor y is denoted by $w[|x| + 1 : |x| + |y|]$. A k-factor is a factor of length k.

A circular string of length m can be viewed as a traditional linear string which has the left-most and right-most symbols wrapped around and stuck together in some way. Under this notion, the same circular string can be seen as m different linear strings, which would all be considered equivalent. Given a string $\mathcal{P}$ of length m, denoted by $\mathcal{P}^i = \mathcal{P}[i:m]\mathcal{P}[1:i-1]$, $0 < i < m$, the i-th *rotation* of $\mathcal{P}$ and $\mathcal{P}^0 = \mathcal{P}$. Consider, for instance, the string $\mathcal{P} = \mathcal{P}^0 = ababbbc$; this string has the following rotations: $\mathcal{P}^1 = bababbca, \mathcal{P}^2 = ababbcab, \mathcal{P}^3 = babbcaba, \mathcal{P}^4 = abbcabab, \mathcal{P}^5 = bbcababa, \mathcal{P}^6 = bcababab, \mathcal{P}^7 = cababbb$.

Here let us consider the problem of finding occurrences of a pattern string $\mathcal{P}$ of length m with circular structure in a text string $\mathcal{T}$ of length n with linear structure. For instance, the DNA sequence of many viruses have a circular structure. So if a biologist wishes to find occurrences of a particular virus in a carrier's DNA sequence, which may not be circular, they must locate all positions in $\mathcal{T}$ where at least one rotation of $\mathcal{P}$ occurs. This is the problem of *circular pattern matching* (CPM).

In the filtering techniques I consider the DNA alphabet, i.e., $\sum = \{A, C, G, T\}$. For the filtering techniques, I represent circular pattern string consisting of these four characters in a way that each character assigned to a numeric value of 1 to 4 for A, C, G and T, respectively, i.e, $A = 1$, $C = 2$, $G = 3$ and $T = 4$. Let us define all the

numeric representation of each character by $i\_distance(\mathcal{P}[`x'])$, where $x \in \sum$. So, $i\_distance(\mathcal{P}[`A']) = 1$, $i\_distance(\mathcal{P}[`C']) = 2$, $i\_distance(\mathcal{P}[`G']) = 3$ and $i\_distance(\mathcal{P}[`T']) = 4$. And all individual distances over the whole string of $\mathcal{P}$ is defined by $i\_distance(\mathcal{P})$. Again, all individual distances over a circular string of $\mathcal{P}$, i.e, $\mathcal{C}(\mathcal{P})$ is defined by $i\_distance(\mathcal{C}(\mathcal{P}))$. I also denote the numeric representation of a string $\mathcal{P}$ by $\mathcal{P}_N$. Though I have only considered DNA alphabets, the numeric representation can be used for all characters $\in \sum^+$. It is called indexing of alphabet.

Example 3 *Suppose I have a pattern $\mathcal{P} = atcgatg$. The numeric representation of pattern $\mathcal{P}_N = 1423143$. And this numeric representation has the following rotations: $\mathcal{P}_N^1 = 4231431$, $\mathcal{P}_N^2 = 2314314$, $\mathcal{P}_N^3 = 3143142$, $\mathcal{P}_N^4 = 1431423$, $\mathcal{P}_N^5 = 4314231$, $\mathcal{P}_N^6 = 3142314$.*

Let us consider two strings, $\mathcal{P}$ of length m and $\mathcal{T}$ of length n, where $n \geq m$. P is said to *circularly match* $\mathcal{T}$ at position i, or equivalently, $\mathcal{C}(\mathcal{P})$ is said to *match* $\mathcal{T}$ at position i, if any circular shift (i.e., conjugates) of $\mathcal{P}$ matches $\mathcal{T}$ at position i of $\mathcal{T}$. Lets define $sum(\mathcal{P})$ by summing up the numeric values of $\mathcal{P}$, i.e, $sum(i\_distance(P))$ in example 3. Here $sum(\mathcal{P}^0) = sum(\mathcal{P}^1) = \cdots = sum(\mathcal{P}^6) = 18$. Summing up over a circular string of $\mathcal{P}$, i.e, $\mathcal{C}(\mathcal{P})$ is defined by $sum(\mathcal{C}(\mathcal{P}))$. It is clear that $sum(\mathcal{P}) = sum(\mathcal{C}(\mathcal{P}))$.

A distance between two consecutive characters of a string $\mathcal{P}$ of length m is defined by $distance(\mathcal{P}[i], \mathcal{P}[i+1])$, where $1 \leq i \leq m-1$ and $distance(\mathcal{P}[i], \mathcal{P}[i+1])$ is the difference between the numeric values of $\mathcal{P}[i]$ and $\mathcal{P}[i+1]$, i.e, $\mathcal{P}[i] - \mathcal{P}[i+1]$. And all distances over the whole string of $\mathcal{P}$ is defined by $distance(\mathcal{P})$. Again, all distances over a circular string of $\mathcal{P}$, i.e, $\mathcal{C}(\mathcal{P})$ is defined by $distance(\mathcal{C}(\mathcal{P}))$. Consider Example 3. Here $distance(\mathcal{P}[1], \mathcal{P}[2]) = 1 - 4 = -3$, $distance(\mathcal{P}[2], \mathcal{P}[3]) = 4 - 2 = 2$ and so on. A distance is referred to as the absolute distance if I take only the magnitude of the difference. I will denote it by $abs(distance(\mathcal{P}[i], \mathcal{P}[i+1]))$. In the same example, $abs(distance(\mathcal{P}[1], \mathcal{P}[2])) = abs(-3) = 3$.

A modulo operation between two consecutive characters of a string $\mathcal{P}$ of length m is defined by $modulo(\mathcal{P}[i], \mathcal{P}[i+1])$, where $1 \leq i \leq m-1$ and $modulo(\mathcal{P}[i], \mathcal{P}[i+1])$ is the modulo operation between the numeric values of $\mathcal{P}[i]$ and $\mathcal{P}[i+1]$, i.e, $\mathcal{P}[i]\%\mathcal{P}[i+1]$. And all modulo operations over the whole string of $\mathcal{P}$ is defined by $modulo(\mathcal{P})$. Again, all modulo operations over a circular string of $\mathcal{P}$, i.e, $\mathcal{C}(\mathcal{P})$ is defined by $modulo(\mathcal{C}(\mathcal{P}))$. In Example 3, $modulo(\mathcal{P}[1], \mathcal{P}[2]) = 1\%4 = 1$, $modulo(\mathcal{P}[2], \mathcal{P}[3]) = 4\%2 = 0$ and so on.

A bitwise exclusive-OR (XOR) operation between two consecutive characters of a string $\mathcal{P}$ of length m is defined by $xor(\mathcal{P}[i], \mathcal{P}[i+1])$, where $1 \leq i \leq m-1$ and $xor(\mathcal{P}[i], \mathcal{P}[i+1])$ is the XOR operation between the numeric values of $\mathcal{P}[i]$ and $\mathcal{P}[i+1]$ i.e, $\mathcal{P}[i] \wedge \mathcal{P}[i+1]$. And all XOR operations over the whole string of $\mathcal{P}$ is defined by $xor(\mathcal{P})$. Again, all XOR operations over a circular string of $\mathcal{P}$, i.e, $\mathcal{C}(\mathcal{P})$ is defined by $xor(\mathcal{C}(\mathcal{P}))$. In Example 3, $xor(\mathcal{P}[1], \mathcal{P}[2]) = 1 \wedge 4 = 5$, $xor(\mathcal{P}[2], \mathcal{P}[3]) = 4 \wedge 2 = 6$ and so on.

Let $\mathcal{P}$ be a non-empty string of length m and $\mathcal{T}$ be a string of length n where $n > m$. Let us say that there exists an occurrence of $\mathcal{P}$ in $\mathcal{T}$, or, more simply, that $\mathcal{P}$ occurs in $\mathcal{T}$, when $\mathcal{P}$ is a factor of $\mathcal{T}$. Every occurrence of P can be characterised by a position in $\mathcal{T}$. Thus we say that $\mathcal{P}$ occurs at the starting position i in $\mathcal{T}$ when $\mathcal{T}[i \ldots i+m-1] = \mathcal{P}$. The *Hamming distance* between strings $\mathcal{P}$ and $\mathcal{T}$, both of length n, is the number of positions i, $0 \leq i < n$, such that $\mathcal{P}[i] \neq \mathcal{T}[i]$. Given a non-negative integer k, write $\mathcal{P} \equiv_k \mathcal{T}$ if the Hamming distance between P and T is at most k.

In this chapter, I consider the following problem.

Problem 1 *(Exact Circular Pattern Matching (ECPM)). Given a pattern $\mathcal{P}$ of length m and a text $\mathcal{T}$ of length $n > m$, find all factors $\mathcal{F}$ of $\mathcal{T}$ such that $\mathcal{F} = \mathcal{P}^i$, $0 \leq i < m$.*

2.3 Filtering algorithm

As it has been mentioned above, my algorithm is based on a number of filtering techniques. Suppose I am given a pattern $\mathcal{P}$ and a text $\mathcal{T}$. I will use the expression "$\mathcal{C}(\mathcal{P})$ matches $\mathcal{T}$ at position i" to indicate that one of the conjugates of $\mathcal{P}$ matches $\mathcal{T}$ at position i. I employ the filters and identify the set $\mathcal{N}$ of indexes of $\mathcal{T}$ such that $\mathcal{C}(\mathcal{P})$ matches at position $i \in \mathcal{N}$. It is clear that for all $j \notin \mathcal{N}$, I do not have a match. And also note that this does not mean that for all $i \in \mathcal{N}$, I will have a match; I may have false positives. So, after I have computed $\mathcal{N}$, I compute $\mathcal{T}'$ which is a reduced version of $\mathcal{T}$. On the reduced text $\mathcal{T}'$ I will employ the average-case algorithms of [4] to get the actual occurrences (the details are given in a later section). To reduce the text string $\mathcal{T}$, I have showed six filtering techniques in [9, 3]. I have slightly changed observations 2, 3, 5 and 6 showed in [9, 3] how to get a better result. To get a better result corresponding to observations 2, 3, 5 & 6 of [9, 3] I do a representation of circular string $\mathcal{P}$ of length m, i.e, $\mathcal{C}'(\mathcal{P}) = \mathcal{P}[1:m]\mathcal{P}[1]$, only wrapping the first character of $\mathcal{P}$ in the last position of the string $\mathcal{P}$ to make the cycle complete. Again while searching the pattern $\mathcal{P}$ of length m in the text string $\mathcal{T}$ of length n where $n \geq m$, I do the same wrapping in the text string in same way that the $\mathcal{T}[i]$-th character is being wrapped in the position $\mathcal{T}[i+m]$ to make the cycle complete. This is done in algorithmic calculation in pattern, not in the actual text. So, this will never effect the subsequent string matching of $\mathcal{P}$ in $\mathcal{T}$ for sliding window calculation. These simple observations are as follows:

Observation 1 *If a string $\mathcal{A}$ circularly matches a string $\mathcal{B}$, then I must have*

$$sum(\mathcal{C}(\mathcal{A})) = sum(\mathcal{B}).$$

.

Example 4 *Suppose I have a pattern $\mathcal{P} = atcgatg$. The numeric representation of pattern $\mathcal{P}_N = 1423143$. And this numeric representation has the following rotations: $\mathcal{P}_N^1 = 4231431$,*

$\mathcal{P}_N^2 = 2314314$, $\mathcal{P}_N^3 = 3143142$, $\mathcal{P}_N^4 = 1431423$, $\mathcal{P}_N^5 = 4314231$. *The text* $\mathcal{T} = tgatcga$ *of the same length, where* $\mathcal{P}$ *circularly matches* $\mathcal{T}$. *The numeric representation of text is* $\mathcal{T}_N = 4314231$. *Here,* $sum(\mathcal{C}(\mathcal{P})) = sum(\mathcal{T}) = 18$.

Observation 2 *If a string* $\mathcal{A}$ *matches a string* $\mathcal{B}$ *by completing the cycle of each string, then I must have*

$$sum(abs(distance(\mathcal{B}))) = sum(abs(distance(\mathcal{C}(\mathcal{A})))).$$

.

Example 5 *Consider the pattern string of* $Example\ 4$. *And a text* $\mathcal{T} = tgatcga$ *of the same length, where* $\mathcal{P}$ *circularly matches* $\mathcal{T}$. *The numeric representation of text* $\mathcal{T}$ *is* $\mathcal{T}_N = 4314231$ *and by completing the cycle the representation is* $\mathcal{T}_N = 43142314$. *Now* $\mathcal{C}'(\mathcal{P})_N = 14231431$. *Here,*
$sum(abs(distance(\mathcal{T}))) = 14$ *and* $sum(abs(distance(\mathcal{C}'(\mathcal{P})))) = 14$. *It is clear that* $sum(abs(distance(\mathcal{T}))) = sum(abs(distance(\mathcal{C}(\mathcal{P}))))$.

Observation 3 *If a string* $\mathcal{A}$ *matches a string* $\mathcal{B}$ *by completing the cycle of each string, then I must have*

$$sum(distance(\mathcal{B})) = sum(distance(\mathcal{C}(\mathcal{A}))).$$

Example 6 *Consider the pattern string of* $Example\ 4$. *And a text* $\mathcal{T} = tgatcga$ *of the same length, where* $\mathcal{P}$ *circularly matches* $\mathcal{T}$. *The numeric representation of text* $\mathcal{T}$ *is* $\mathcal{T}_N = 4314231$ *and by completing the cycle the representation is* $\mathcal{T}_N = 43142314$. *Now* $\mathcal{C}'(\mathcal{P})_N = 14231431$. *Here,*
$sum(distance(\mathcal{T})) = 0$ *and* $sum(distance(\mathcal{C}'(\mathcal{P}))) = 0$. *It is clear that* $sum(distance(\mathcal{T})) = sum(distance(\mathcal{C}(\mathcal{P})))$.

Observation 4 *If a string* $\mathcal{A}$ *circularly matches a string* $\mathcal{B}$, *then I must have*

$$sum(i\_distance(\mathcal{C}(\mathcal{A}))) = sum(i\_distance(\mathcal{B})).$$

Example 7 *Consider Example 4. Here all individual distances over the whole string of* $\mathcal{C}(\mathcal{P})$ *are* $sum(i\_distance(`A')) = 2$, $sum(i\_distance(`C')) = 2$, $sum(i\_distance(`G')) = 6$ *and* $sum(i\_distance(`T')) = 8$. *And here all individual distances over the whole string of* $\mathcal{T}$ *are* $sum(i\_distance(`A')) = 2$, $sum(i\_distance(`C')) = 2$, $sum(i\_distance(`G')) = 6$ *and* $sum(i\_distance(`T')) = 8$. *It is clear that,* $sum(i\_distance(\mathcal{C}(\mathcal{P}))) = sum(i\_distance(\mathcal{T}))$.

Observation 5 *If a string* $\mathcal{A}$ *matches a string* $\mathcal{B}$ *by completing the cycle of each string, then I must have*

$$sum(modulo(\mathcal{B})) = sum(modulo(\mathcal{C}(\mathcal{A}))).$$

Example 8 *Consider the pattern string of Example 4. And a text* $\mathcal{T} = tgatcga$ *of the same length, where* $\mathcal{P}$ *circularly matches* $\mathcal{T}$. *The numeric representation of text* $\mathcal{T}$ *is* $\mathcal{T}_N = 4314231$ *and by completing the cycle the representation is* $\mathcal{T}_N = 43142314$. *Now* $\mathcal{C}'(\mathcal{P})_N = 14231431$. *Here,*

$sum(modulo(\mathcal{T})) = 5$ *and* $sum(modulo(\mathcal{C}'(\mathcal{P}))) = 5$. *It is clear that*

$sum(modulo(\mathcal{T})) = sum(modulo(\mathcal{C}'(\mathcal{P})))$.

Observation 6 *If a string* $\mathcal{A}$ *matches a string* $\mathcal{B}$ *by completing the cycle of each string, then I must have*

$$sum(xor(\mathcal{B})) = sum(xor(\mathcal{C}(\mathcal{A}))).$$

Example 9 *Consider the pattern string of Example 4. And a text* $\mathcal{T} = tgatcga$ *of the same length, where* $\mathcal{P}$ *circularly matches* $\mathcal{T}$. *The numeric representation of text* $\mathcal{T}$ *is* $\mathcal{T}_N = 4314231$ *and by completing the cycle the representation is* $\mathcal{T}_N = 43142314$. *Now* $\mathcal{C}'(\mathcal{P})_N = 14231431$. *Here,*

$sum(xor(\mathcal{T})) = 28$ *and* $sum(xor(\mathcal{C}'(\mathcal{P}))) = 28$. *It is clear that*

$sum(xor(\mathcal{T})) = sum(xor(\mathcal{C}'(\mathcal{P})))$.

Example 10 *Suppose I have a text* $\mathcal{T} = tgatcgaaagta$ *of length* $n = 12$ *and consider Example 3. Here the length of the pattern* $\mathcal{P}$ *is* 7, *i.e,* $m = 7$. *The numeric representation of text* $\mathcal{T}_N = 431423111341$. *Now, Apply observations* $1 : 6$ *between* $\mathcal{P}$ *and* $\mathcal{T}[1 : m]$.

For observations 2, 3, 5 *and* 6 *I need to wrap the first starting character of pattern* $\mathcal{P}$ *and text* $\mathcal{T}$ *to complete the cycle like the examples* 3, 4, 6 *and* 7, *respectively. Follow* $Examples$ $4 : 9$ *in that case. It is the* $first$ *iteration. According to the* $Example$ $2 : 7$ *I have a filtered match but not the exact match. For the next iterations, apply observation* $1 : 6$ *between* $\mathcal{P}$ *and* $\mathcal{T}[2 : m + 1]$; $\mathcal{P}$ *and* $\mathcal{T}[3 : m + 2]$; $\mathcal{P}$ *and* $\mathcal{T}[4 : m + 2]$ *and so on exactly* $n - m = 12 - 7 = 5$ *times. In each iteration, do just like* $Examples$ $4 : 9$. *Note that, remove the calculation of first wrapped character before starting the next iteration from* $\mathcal{T}$ *that was added to complete the cycle for observations* 2, 3, 5 *and* 6. *This is left as an exercise for the reader.*

2.3.1 Circular pattern signature using filtering techniques

In this subsection, I present an $\mathcal{O}(m)$ time algorithm to compute the signature of a pattern of length m. This Signature will be used later to filter the text. Here, I need five variables to save the filtered values based on *Observations* 1,2, 3, 5 and 6. And to save the values of *Observation* 4, I need a list of size 4. I start with the string $\mathcal{P}[1 : m]\mathcal{P}[1]$, i.e., the concatenation of $\mathcal{P}[1 : m]$ and $\mathcal{P}[1]$. This is due to complete the cycle in pattern $\mathcal{P}$. Now, I calculate the values according to *Observations* 1 to 6 for $\mathcal{P}[1 : m]$. This iterates m times to get a single value for each observation. Overall runtime of the algorithm is $\mathcal{O}(m)$. *Algorithm* 1, $ECPS\_FT$ gives the corresponding pseudo code.

2.3.2 Reduction of search space in the text

I have presented an $\mathcal{O}(n)$ runtime algorithm to reduce the search space of the text string for exact CPM in [3]. Using the same algorithm presented in [3] I present an $\mathcal{O}(n)$ runtime algorithm to reduce the search space of the text applying the filtering discussed above. In this algorithm I use the output of Algorithm $ECPS\_FT$. I follow the same technique that I used in $Algorithm$ ECPS in [3]. Here I calculate values according to *Observations* $1 : 6$ for $T[1 : m]$ and wrapping the first $T[1]$ character in $T[m + 1]$-th position to make cycle complete. This iterates $m + 1$ times to get a single

Algorithm 1 Exact Circular Pattern Signature using Observations $1:6$ in a single pass

procedure $ECPS\_FT(\mathcal{P}[1:m])$
    define five varibles for observations 1, 2, 3, 5, 6
    define an array of size 4 for observation 4
    define an array of size 4 to keep fixed value of A, C, G, T
    $s \leftarrow \mathcal{P}[1:m]\mathcal{P}[1]$
    initialize all defined variables to zero
    initialize fixed array to $\{1, 2, 3, 4\}$
    for $i \leftarrow 1$ to $|s|$ **do**
        if $i \neq |s|$ **then**
            calculate different filtering values via observations 1 & 4 and make a running sum
        end if
        calculate different filtering values via observations 2, 3, 5 & 6 and make a running sum
    end for
    return all observations values
end procedure

value for each observation. Then I check it with the returned values of $ECPS\_FT$ algorithm. If it matches, then I output the text string $\mathcal{T}[1:m]$ to a file. After that, I apply a sliding window approach as follows: I apply the same technique to calculate the subsequent values by subtracting the value of the i-th character along with the wrapped one and adding the same of the $(i+m)$-th character and also adding the new wrapping value of $i+1$-th character. This also iterates exactly $n-m$ times to get other $n-m$ values of *Observations* 1 to 6 and output the text string to file if it matches to all observations in same way in each iteration. Overall runtime of the algorithm is $\mathcal{O}(m) + \mathcal{O}(n-m) = \mathcal{O}(n)$. *Algorithm* RSS_FT in [3] gives the corresponding pseudo code.

2.3.3 Fast ECPM algorithm

In [4], the authors have presented an approximate circular string matching with k-mismatches via filtering (ACSMF-Simple). They have built a library for ACSMF-Simple algorithm. The library is freely available and can be found here: [13]. In

Algorithm 2 Exact Circular Pattern Matching (ECPM) using procedure ECPS_FT

procedure $ECPM\_FT(\mathcal{T}[1:n], \mathcal{P}[1:m])$
  CALL ECPS_FT($\mathcal{P}[1:m]$)
  save the return value of observations $1:6$ for further use here
  define an array of size 4 to keep fixed value of A, C, G, T
  initialize fixed array to $\{1, 2, 3, 4\}$
  for $i \leftarrow 1$ to m **do**
    calculate different filtering values in $\mathcal{T}[i:m]$ via observations $1:6$ and make a running sum
  end for
  if $1:6$ observations values of $\mathcal{P}[1:m]$ vs $1:6$ observations values of $\mathcal{T}[1:m]$ have a match for the calculated ranges of each filters **then**
    ▷ Found a filtered match
    Found a candidate of $\mathcal{T}'$
    Run ACSMF-SimpleZero$k$ over this candidate of $\mathcal{T}'$ and report the position accordingly if a match found
  end if
  for $i \leftarrow 1$ to $n - m$ **do**
    calculate different filtering values in $\mathcal{T}[i:m]$ via observations $1:6$ by subtracting i-th value and wrapped value and adding $i+m$-th value to the running sum and also adding the newly wrapped value to complete the cycle
    if $1:6$ filtering values of $\mathcal{P}[1:m]$ vs $1:6$ filtering values of $\mathcal{T}[i+1:i+m]$ have a match **then**
      ▷ Found a filtered match
      check whether non-consecutive sequence or not. if non-consecutive, report as another candidate of $\mathcal{T}'$
      Run ACSMF-SimpleZero$k$ over this candidate of $\mathcal{T}'$ and report the position accordingly if a match found
    end if
  end for
end procedure

this algorithm, if I use $k = 0$, then ACSMF-Simple works for exact case. In what follows, I will refer to this algorithm as ACSMF-SimpleZero$k$. I have implemented a fast ECPM algorithm using ACSMF-Simple algorithm simply by putting $k = 0$. *Algorithm* $ECPM\_FT$ gives the corresponding pseudo code, where I need to call ACSMF-Simple algorithm with $k = 0$. In what follows, I will refer to this algorithm as Filter-ECPM. It is clear that, the runtime of Filter-ECPM is $O(n)$ in the average case.

2.4 Experimental results

I have implemented the modified Filter-ECPM algorithm using the ACSMF-Simple [4]. In [4], ACSMF-Simple were implemented as library functions in the *C* programming language under *GNU/Linux* operating system. The library implementation is distributed under the GNU General Public License (GPL). ACSMF-Simple takes as input the pattern $\mathcal{P}$ of length m, the text $\mathcal{T}$ of length n, and the integer threshold $k < m$ and returns the list of starting positions of the occurrences of the rotations of $\mathcal{P}$ in $\mathcal{T}$ with k-mismatches as output.

I have used real genome data as experimental text string, $\mathcal{T}$. I have collected that data from [62]. Here, I have taken *299MB* of data for our experiment. I have generated random patterns of different length by a random indexing technique in this *299MB* of text string.

The experiments of [4] were conducted on a Desktop PC using one core of Intel *i7 2600 CPU at 3.4 GHz* under GNU/Linux. But, I used the library of [4] on a PowerEdge R820 rack serve PC with 6-core of Intel Xeon processor *E5-4600* product family and 64GB of RAM under GNU/Linux. With the help of the library used in [4], I compared the running time of Filter-ECPM [3] and ACSMF-SimpleZero$k$. Again, I compared the running time of modified Filter-ECPM and ACSMF-SimpleZero$k$. Table 2.1 reports the elapsed time and speed-up comparisons for various pattern sizes $(500 \leq m \leq 3000)$. As can be seen from Table 2.1, Algorithm Filter-ACPM [3] runs

TABLE 2.1: Elapsed-time and speed-up comparisons of Filter-ECPM[3], ACSMF-SimpleZero$k$ and modified Filter-ECPM for text $n = 299MB$

m	Elapsed Time(s) of ACSMF-simpleZero$k$	Elapsed Time(s) of Filter-ECPM [3]	Speed up: ACSMF-simpleZero$k$ vs. Filter-ECPM [3]	Elapsed Time(s) of Modified Filter-ECPM	Speed up: ACSMF-simpleZero$k$ vs. Modified Filter-ECPM
500	5.938	3.025	2	1.167	5
550	7.914	3.068	3	1.456	5
600	7.691	3.06	3	1.364	6
650	7.836	3.074	3	1.006	8
700	7.739	3.072	3	1.028	8
750	7.82	3.051	3	1.073	7
800	7.839	3.209	2	1.04	8
850	8.382	3.053	3	1.055	8
900	7.646	3.055	3	1.278	6
950	7.876	3.049	3	1.402	6
1000	7.731	3.067	3	1.216	6
1600	7.334	3.206	2	1.182	6
1650	8.239	3.063	3	0.969	9
1700	7.572	3.059	2	1.18	6
1750	5.968	3.066	2	1.144	5
1800	7.551	3.064	2	1.179	6
1850	7.407	3.079	2	1.086	7
1900	7.861	3.225	2	1.126	7
1950	7.339	3.073	2	1.028	7
2000	7.814	3.062	3	1.118	7
2050	5.969	3.057	2	1.988	3
2100	5.173	3.036	2	1.187	4
2150	5.317	3.027	2	1.919	3
2200	6.032	3.168	2	1.927	3
2250	5.009	3.073	2	1.895	3
2300	5.029	3.024	2	1.891	3
2350	5.041	3.047	2	1.887	3
2400	6.036	3.046	2	1.91	3
2450	6.04	3.037	2	1.886	3
2500	7.046	3.029	2	1.976	4
2550	7.042	3.037	2	1.987	4
2600	8.043	4.029	2	2.883	3
2650	8.049	4.03	2	2.884	3
2700	8.031	4.183	2	2.892	3
2750	8.039	4.044	2	2.882	3
2800	9.026	4.067	2	2.886	3
2850	9.154	4.036	2	2.901	3
2900	10.049	4.045	2	3.134	3
2950	11.044	5.052	2	3.876	3
3000	12.044	6.039	2	3.9	3

faster than ACSMF-SimpleZero$k$ in all cases. And in fact the Filter-ECPM [3] achieves a minimum of two-fold speed-up for all the pattern sizes. Again, referring to the same table, $Algorithm$ 2, modified Filter-ACPM runs more faster than ACSMF-SimpleZero$k$ in all cases. And in fact the modified Filter-ECPM achieves a minimum of three-fold speed-up for all the pattern sizes. Now, I can summarize that, modified Filter-ECPM of this paper, $Algortihm$ 2 is more faster than that of Filter-ECPM [3]. Note that, I have experimented on same set of data, same input pattern and so forth. The program source code link is available at 9.

Chapter 3

Aproximate Circular Pattern Matching

3.1 Introduction

Earlier, I briefly described that the circular pattern, denoted $\mathcal{C}(\mathcal{P})$, corresponding to a given pattern $\mathcal{P} = \mathcal{P}_1 \ldots \mathcal{P}_m$, is formed by connecting $\mathcal{P}_1$ with $\mathcal{P}_m$ and forming a sort of a cycle; this gives us the notion where the same circular pattern can be seen as m different linear patterns, which would all be considered equivalent.

In this chapter I focus on the Approximate Circular Pattern Matching (ACPM) problem. As it has been mentioned earlier in literature review, the DNA sequence of many viruses have circular structures. So if a biologist wishes to find occurrences of a particular virus in a carrier's (linear) DNA sequence, (s)he must locate all positions in $\mathcal{T}$ where at least one rotation of $\mathcal{P}$ occurs. This motivates one to study CPM. Now, the comment made in the introduction regarding the suitability of allowing errors in locating occurrences (i.e., approximate version) rather than considering exact occurrences (i.e., exact version) remains true in the context of circular pattern matching as well. Therefore, the biologists are more interested in locating the approximate occurrences of one of the rotations of $\mathcal{P}$ in $\mathcal{T}$, i.e., they like to allow errors while locating the occurrences. In a practical context, such errors may arise due to various reasons

ranging from natural mutations in the DNA of the virus to the practical limitations of the lab equipments that may introduce errors while sequencing. This motivates the study of ACPM i.e., the approximate version of the problem.

The rest of the chapter is organized as follows. Section 3.2 gives a preliminary description and definition of some terminologies and concepts related to stringology that will be used throughout this chapter. In Section 3.3 I describe the filtering algorithms. Section 3.4 presents the experimental results.

3.2 Problem definition

The basics of circular string have been described in the previous chapter.

Similar to Chapter 2, DNA alphabet, i.e., $\Sigma = \{a, c, g, t\}$ is considered for my algorithms and implementation for approximate circular pattern matching in this chapter. The approach is, each character of the alphabet is associated to a numeric value as follows. Each character is assigned a unique numbers from the range $[1...|\Sigma|]$. Although this is not essential, I conveniently assign the numbers from the range $[1...|\Sigma|]$ to the characters of Σ following their inherent lexicographical order. I use $num(x), x \in \Sigma$ to denote the numeric value of the character x. So, I have $num(a) = 1, num(c) = 2, num(g) = 3$ and $num(t) = 4$,. For a string S, I use the notation S_N to denote the numeric representation of the string S; and $S_N[i]$ denotes the numeric value of the character $S[i]$. So, if $S[i] = g$ then $S_N[i] = num(g) = 3$. The concept of circular string and their rotations also apply on their numeric representations as is illustrated in Example 1 below.

Example 1 *Suppose I have a pattern* $\mathcal{P} = atcgatg$. *The numeric representation of* $\mathcal{P}$ *is* $\mathcal{P}_N = 1423143$. *And this numeric representation has the following rotations:* $\mathcal{P}_N^1 = 4231431$, $\mathcal{P}_N^2 = 2314314$, $\mathcal{P}_N^3 = 3143142$, $\mathcal{P}_N^4 = 1431423$, $\mathcal{P}_N^5 = 4314231$, $\mathcal{P}_N^6 = 3142314$.

The problem I handle in this chapter can be formally defined as follows.

Problem 1 *(Approximate Circular Pattern Matching with k-mismatches (i.e mutations) (ACPM)). Given a pattern $\mathcal{P}$ of length m, a text $\mathcal{T}$ of length $n > m$, and an integer threshold $k < m$, find all factors $\mathcal{F}$ of $\mathcal{T}$ such that $\mathcal{F} \equiv_k \mathcal{P}^i$ for some $0 \leq i < m$. And when I have a factor $\mathcal{F} = \mathcal{T}[j : j + |\mathcal{F}| - 1]$ such that $\mathcal{F} \equiv_k \mathcal{P}^i$ lets say that the circular pattern $\mathcal{C}(\mathcal{P})$ k-matches $\mathcal{T}$ at position j. I also say that this k-match is due to $\mathcal{P}^i$, i.e., the ith rotation of $\mathcal{P}$.*

In the context of the filter based algorithms the concept of false positives and negatives is important. Suppose I have an algorithm $\mathcal{A}$ to solve a problem $\mathcal{B}$. Now suppose that $\mathcal{S}_{true}$ represents the set of true solutions for the problem $\mathcal{B}$. Further suppose that $\mathcal{A}$ computes the set $\mathcal{S}_{\mathcal{A}}$ as the set of solutions for $\mathcal{B}$. Now assume that $\mathcal{S}_{true} \neq \mathcal{S}_{\mathcal{A}}$. Then, the set of false positives can be computed as follows: $\mathcal{S}_{\mathcal{A}} \setminus \mathcal{S}_{true}$. In other words, the set computed by $\mathcal{A}$ contains some solutions that are not true solutions for problem $\mathcal{B}$. And these are the false positives, because, $\mathcal{S}_{\mathcal{A}}$ falsely marked these as solutions (i.e., positive). On the other hand, the set of false negatives can be computed as follows: $\mathcal{S}_{true} \setminus \mathcal{S}_{\mathcal{A}}$. In other words, false negatives are those members in $\mathcal{S}_{true}$ that are absent in $\mathcal{S}_{\mathcal{A}}$. These are false negatives because $\mathcal{S}_{\mathcal{A}}$ falsely marked these as non-solutions (i.e., negative).

3.3 Filtering Algorithm

As it has been mentioned above, the algorithm is based on some filtering techniques. In particular I will be extending the filters used by SimpLiFiCPM [10] to make it useful and effective in the context of approximate circular pattern matching. In what follows, I follow the notations of [10]. Suppose I am given a pattern $\mathcal{P}$ and a text $\mathcal{T}$. We will frequently and conveniently use the expression "$\mathcal{C}(\mathcal{P})$ k-matches $\mathcal{T}$ at position i" (or equivalently, "$\mathcal{P}$ circularly k-matches $\mathcal{T}$ at position i") to indicate that one of the conjugates of $\mathcal{P}$ k-matches $\mathcal{T}$ at position i (or equivalently, $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$). I start with a brief overview of the approach below.

3.3.1 Overview of SimpLiFiACPM

The approach follows recent work in [9, 11] where I used number of filters to solve the both exact and approximate circular pattern matching problem. In particular we extended the ideas of [9, 11] and adapt the filters presented there so that those filters become useful and effective for the approximate version as well.

3.3.2 Filters

I employ a total of 6 filters. I have run this experiment on three candidate patterns of different length. As there could be 6!, i.e., 720 combinations, I have recorded the running times of the algorithm for all these combinations considering the three candidate patterns. Although it was not possible to identify one ordering that performs best in all instances, but it appeared that applying filter 4 first produces best result in all instances. The results are available in an URL given in Appendix section9. The key to the observations and the resulting filters is the fact that each function devises results in a unique output when applied to the rotations of a circular string. For example, consider a hypothetical function $\mathcal{X}$. I will always have the relation that $\mathcal{X}(\mathcal{P}) = \mathcal{X}(\mathcal{P}^i)$ for all $1 \leq i < n$. Recall that, $\mathcal{P}^0$ actually denotes $\mathcal{P}$. For the sake of conciseness, for such functions, I will manipulate the notation a bit and use $\mathcal{X}(\mathcal{C}(\mathcal{P}))$ to represent $\mathcal{X}(\mathcal{P}^i)$ for all $0 \leq i < |\mathcal{P}|$.

Filter 1

I define the function $sum()$ on a string $\mathcal{P}$ of length m as follows: $sum(\mathcal{P}) = \sum_{i=1}^{m} P_N[i]$. My first filter, Filter 1, is based on this $sum()$ function. I have the following observation.

Observation 1 *Consider a circular string $\mathcal{P}$ and a linear string $\mathcal{T}$ both having length n. If $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$, where $0 \leq k < n$, then I must have*

$$sum(\mathcal{T}) - k \times 4 + k \times 1 \leq sum(\mathcal{C}(\mathcal{P})) \leq sum(\mathcal{T}) + k \times 4 - k \times 1.$$

Example 2 *Consider $\mathcal{P} = atcgatg$. I can easily calculate that $sum(\mathcal{C}(\mathcal{P})) = 18$. Now, consider $\mathcal{T}1 = aacgatg$, slightly different from $\mathcal{P}$, i.e, $\mathcal{P}[2] = t \neq \mathcal{T}1[2] = a$. As can be easily verified, here $\mathcal{P} \equiv_1 \mathcal{T}1$. According to Observation 1, in this case the lower (upper) bound is 15 (18). Indeed, I have ${\mathcal{T}1}_N = 1123143$ and $sum(\mathcal{T}1) = 15$, which is within the bounds. Now consider $\mathcal{T}2 = ttcgatg$, slightly different from $\mathcal{P}$, i.e, $\mathcal{P}[1] = a \neq \mathcal{T}2[1] = t$. As can be easily verified, here $\mathcal{P} \equiv_1 \mathcal{T}2$. Therefore, in this case as well, the lower and upper bound mentioned above hold. And indeed I have ${\mathcal{T}2}_N = 4423143$ and $sum(\mathcal{T}2) = 21$, which is within the bounds. Finally, consider another string $\mathcal{T}' = atagctg$. It can be easily verified that $\mathcal{C}(\mathcal{P}) \not\equiv_1 \mathcal{T}'$. Again, the previous bounds hold in this case and I find that ${\mathcal{T}'}_N = 1413243$ and $sum(\mathcal{T}') = 18$. Clearly this is within the bounds of Observation 1 and in fact it is exactly equal to $sum(\mathcal{C}(\mathcal{P}))$. This is an example of a false positive with respect to Filter 1.*

Filters 2 and 3

The second and third filters, i.e., Filters 2 and 3, depend on a notion of distance between consecutive characters of a string. The *distance* between two consecutive characters of a string $\mathcal{P}$ of length m is defined by $distance(\mathcal{P}[i], \mathcal{P}[i+1]) = \mathcal{P}_{\mathcal{N}}[i] - \mathcal{P}_{\mathcal{N}}[i+1]$, where $1 \leq i \leq m-1$. I define $total\_distance(P) = \sum_{i=1}^{m-1} distance(\mathcal{P}[i], \mathcal{P}[i+1])$. I also define an absolute version of it: $abs\_total\_distance(P) = \sum_{i=1}^{m-1} abs(distance(\mathcal{P}[i], \mathcal{P}[i+1]))$, where $abs(x)$ returns the magnitude of x ignoring the sign. Before applying these two functions on the strings to get the filters, I need to do a simple pre-processing on the respective string, i.e., $\mathcal{P}$ in this case as follows. I extend the string $\mathcal{P}$ by concatenating the first character of $\mathcal{P}$ at its end. I use $ext(\mathcal{P})$ to denote the resultant string. So, I have $ext(\mathcal{P}) = \mathcal{P}\mathcal{P}[1]$. Since, $ext(\mathcal{P})$ can simply be treated as another string, I can easily

extend the notation and concept of $\mathcal{C}(\mathcal{P})$ over $ext(\mathcal{P})$ and I continue to abuse the notation a bit for the sake of conciseness as mentioned at the beginning of Section 3.3.2 (just before Section 3.3.2).

Now I have the following observation which is the basis of the Filter 2.

Observation 2 *Consider a circular string $\mathcal{P}$ and a linear string $\mathcal{T}$ both having length n and assume that $\mathcal{A} = ext(\mathcal{P})$ and $\mathcal{B} = ext(\mathcal{T})$. If $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$, where $0 \leq k < n$, then I must have*

$$abs\_total\_distance(\mathcal{B}) - k \times 4 + k \times 1 \leq$$

$$abs\_total\_distance(\mathcal{C}(\mathcal{A})) \leq$$

$$abs\_total\_distance(\mathcal{B}) + k \times 4 - k \times 1.$$

Example 3 *Consider the same strings of Example 2, i.e., $\mathcal{P} = atcgatg$, $\mathcal{T}1 = aacgatg$ and $\mathcal{T}2 = ttcgatg$. As can be easily verified, here $\mathcal{P} \equiv_1 \mathcal{T}1$ and $\mathcal{P} \equiv_1 \mathcal{T}2$. Now consider the extended strings and assume that $\mathcal{A} = ext(\mathcal{P})$, $\mathcal{B}1 = ext(\mathcal{T}1)$ and $\mathcal{B}2 = ext(\mathcal{T}2)$. It can be easily verified that $abs\_total\_distance(\mathcal{C}(\mathcal{A}))$ is 14. Recall that $\mathcal{T}1$ is slightly different from $\mathcal{P}$, i.e, $\mathcal{P}[2] = t \neq \mathcal{T}1[2] = a$. Now I have $\mathcal{T}1_N = 1123143$. Hence $\mathcal{B}1_N = 11231431$. Hence, $abs\_total\_distance(\mathcal{B}1) = 10$ which is indeed within the bounds of Observation 2. Now consider $\mathcal{T}2$, which is slightly different from $\mathcal{P}$, i.e, $\mathcal{P}[1] = a \neq \mathcal{T}2[1] = t$. Now I have $\mathcal{T}2_N = 4423143$. Hence $\mathcal{B}2_N = 44231434$. Hence, $abs\_total\_distance(\mathcal{B}2) = 10$, which is also within the bounds. Finally, consider $\mathcal{T}' = atagctg$, which is again slightly different from $\mathcal{P}$. It can be easily verified that $\mathcal{C}(\mathcal{P}) \not\equiv_1 \mathcal{T}'$. However, assuming that $\mathcal{B}' = ext(\mathcal{T}')$ I find that $abs\_total\_distance(\mathcal{B}')$ is still 14, which is in the range of Observation 2. This is an example of a false positive with respect to Filter 2.*

Now I present the following related observation which is the basis of the Filter 3. Note that Observation 2 differs with Observation 3 only through using the absolute version of the function used in the latter.

Observation 3 *Consider a circular string $\mathcal{P}$ and a linear string $\mathcal{T}$ both having length n and assume that $\mathcal{A} = ext(\mathcal{P})$ and $\mathcal{B} = ext(\mathcal{T})$. If $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$, where $0 \leq k < n$, then I must have*

$$total\_distance(\mathcal{B}) - k \times 4 + k \times 1 \leq total\_distance(\mathcal{C}(\mathcal{A}))$$

$$\leq total\_distance(\mathcal{B}) + k \times 4 - k \times 1.$$

Example 4 *Consider the same strings of Example 2, i.e., $\mathcal{P} = atcgatg$, $\mathcal{T}1 = aacgatg$ and $\mathcal{T}2 = ttcgatg$. As can be easily verified, here $\mathcal{P} \equiv_1 \mathcal{T}1$ and $\mathcal{P} \equiv_1 \mathcal{T}2$. Now consider the extended strings and assume that $\mathcal{A} = ext(\mathcal{P})$, $\mathcal{B}1 = ext(\mathcal{T}1)$ and $\mathcal{B}2 = ext(\mathcal{T}2)$. It can be easily verified that $abs\_total\_distance(\mathcal{C}(\mathcal{A}))$ is 14. Recall that $\mathcal{T}1$ is slightly different from $\mathcal{P}$, i.e, $\mathcal{P}[2] = t \neq \mathcal{T}1[2] = a$. Now I have $\mathcal{T}1_N = 1123143$. Hence $\mathcal{B}1_N = 11231431$. Hence, $total\_distance(\mathcal{B}1) = 0$ which is indeed within the bounds of Observation 2. Now consider $\mathcal{T}2$, which is slightly different from $\mathcal{P}$, i.e, $\mathcal{P}[1] = a \neq \mathcal{T}2[1] = t$. Now I have $\mathcal{T}2_N = 4423143$. Hence $\mathcal{B}2_N = 44231434$. Hence, $total\_distance(\mathcal{B}2) = 10$, which is also within the bounds. Finally, consider $\mathcal{T}' = atagctg$, which is again slightly different from $\mathcal{P}$. It can be easily verified that $\mathcal{C}(\mathcal{P}) \not\equiv_1 \mathcal{T}'$. However, assuming that $\mathcal{B}' = ext(\mathcal{T}')$ I find that $total\_distance(\mathcal{B}')$ is still 0, which is in the range of Observation 2. This is an example of a false positive with respect to Filter 3.*

Filter 4

Filter 4 uses the $sum()$ function used by Filter 1, albeit, in a slightly different way. In particular, it applies the $sum()$ function on individual characters. So, for $x \in \Sigma$ I define $sum_x(\mathcal{P}) = \sum_{1 \leq i \leq |\mathcal{P}|, \mathcal{P}[i]=x} P_N[i]$. Now I have the following observation.

Observation 4 *Consider a circular string $\mathcal{P}$ and a linear string $\mathcal{T}$ both having length n. If $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$, where $0 \leq k < n$, then I must have*

$$sum_x(\mathcal{T}) - k \times num(x) \leq sum_x(\mathcal{C}(\mathcal{P}))$$

$$\leq sum_x(\mathcal{T}) + k \times num(x)$$

for all $x \in \Sigma$.

Example 5 *Consider the same strings of Example 2, i.e.,* $\mathcal{P} = atcgatg$, $\mathcal{T}1 = aacgatg$ *and* $\mathcal{T}2 = ttcgatg$. *Recall that here* $\mathcal{P} \equiv_1 \mathcal{T}1$ *and* $\mathcal{P} \equiv_1 \mathcal{T}2$. *Now, as it has been described in Example 2 that* $\mathcal{T}1$ *is slightly different from* $\mathcal{P}$, *i.e,* $\mathcal{P}[2] = t \neq \mathcal{T}1[2] = a$. *Now I have* $\mathcal{T}1_N = 1123143$. *Hence,* $sum_a(\mathcal{T}1) = 3$, $sum_c(\mathcal{T}1) = 2$, $sum_g(\mathcal{T}1) = 6$ *and* $sum_t(\mathcal{T}1) = 4$. *According to Observation 4, in this case the lower (upper) bound for character 'A' is* 2 (3). *And the lower (upper) bound for character 'T' is* 4 (8), *which is in the bounds. Now consider that* $\mathcal{T}2$ *which is slightly from* $\mathcal{P}$, *i.e* $\mathcal{P}[1] = a \neq \mathcal{T}2[1] = t$. *Now I have* $\mathcal{T}2_N = 4423143$. *Hence,* $sum_a(\mathcal{T}2) = 1$, $sum_c(\mathcal{T}2) = 2$, $sum_g(\mathcal{T}2) = 6$ *and* $sum_t(\mathcal{T}1) = 12$. *Here, in this case the lower (upper) bound for character 'A' is* 1 (2). *And the lower (upper) bound for character 'T' is* 8 (12), *which is also in the bounds. Therefore, I can summarize that I have got a lower bound and an upper bound according to Observation 4 for these two characters 'A' and 'T', others are unchanged. For this example the Observation 4 shows the overall lower (upper) bound for character 'A' is* 1 (3) *and the lower (upper) bound for character 'T' is* 4 (12). *Finally, consider* $\mathcal{T}' = atagctg$, *which is again slightly different from* $\mathcal{P}$. *It can be easily verified that* $\mathcal{C}(\mathcal{P}) \not\equiv_1 \mathcal{T}'$. *Now,* $\mathcal{T}'_N = 1413243$. *However, as I can still find that,* $sum_a(\mathcal{T}') = 2$, $sum_c(\mathcal{T}') = 2$, $sum_g(\mathcal{T}') = 6$ *and* $sum_t(\mathcal{T}') = 8$, *which is in the bounds of Observation 4. This is an example of a false positive with respect to Filter 4.*

Filter 5

In Filter 5, I use $modulo()$ operation between two consecutive characters. A $modulo()$ operation between two consecutive characters of a string $\mathcal{P}$ of length m is defined as follows: $modulo(\mathcal{P}[i], \mathcal{P}[i+1]) = \mathcal{P}_N[i] \% \mathcal{P}_N[i+1]$, where $1 \leq i \leq m-1$. We define $sum\_modulo(\mathcal{P})$ to be the summation of the results of the modulo operations on the consecutive characters of $\mathcal{P}$. More formally, $sum\_modulo(P) = \sum_{i=1}^{m-1} modulo(\mathcal{P}[i], \mathcal{P}[i+1])$.

Now I present the following observation which is the basis of Filter 5. Note that this observation is applied on the extended versions of the respective strings.

Observation 5 *Consider a circular string $\mathcal{P}$ and a linear string $\mathcal{T}$ both having length n and assume that $\mathcal{A} = ext(\mathcal{P})$ and $\mathcal{B} = ext(\mathcal{T})$. If $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$, where $0 \leq k < n$, then, I must have*

$$sum\_modulo(\mathcal{B}) - k \times 4 + k \times 0 \leq sum\_modulo(\mathcal{C}(\mathcal{A}))$$

$$\leq sum\_modulo(\mathcal{B}) + k \times 4 - k \times 0.$$

Note carefully that the function $sum\_modulo()$ has been applied on the extended strings.

Example 6 *Consider the same four strings of Example 3, i.e., $\mathcal{P} = atcgatg$, $\mathcal{T}1 = aacgatg$ and $\mathcal{T}2 = ttcgatg$. As can be easily verified, here $\mathcal{P} \equiv_1 \mathcal{T}1$ and $\mathcal{P} \equiv_1 \mathcal{T}2$. Now consider the extended strings and assume that $\mathcal{A} = ext(\mathcal{P})$, $\mathcal{B}1 = ext(\mathcal{T}1)$ and $\mathcal{B}2 = ext(\mathcal{T}2)$. It can be easily verified that $sum\_modulo(\mathcal{C}(\mathcal{A}))$ is 5. Recall that $\mathcal{T}1$ is slightly different from $\mathcal{P}$, i.e, $\mathcal{P}[2] = t \neq \mathcal{T}1[2] = a$. Now I have $\mathcal{T}1_N = 1123143$. Hence $\mathcal{B}1_N = 11231431$. Hence, $sum\_modulo(\mathcal{B}1) = 5$ which is indeed within the bounds of Observation 5. Now consider $\mathcal{T}2$, which is slightly different from $\mathcal{P}$, i.e, $\mathcal{P}[1] = a \neq \mathcal{T}2[1] = t$. Now I have $\mathcal{T}2_N = 4423143$. Hence $\mathcal{B}2_N = 44231434$. Hence, $sum\_modulo(\mathcal{B}2) = 7$, which is also within the bounds. Finally, consider $\mathcal{T}' = atagctg$, which is again slightly different from $\mathcal{P}$. It can be easily verified that $\mathcal{C}(\mathcal{P}) \not\equiv_1 \mathcal{T}'$. However, assuming that $\mathcal{B}' = ext(\mathcal{T}')$ I find that $sum\_modulo(\mathcal{B}')$ is still 5, which is in the range of Observation 5. This is an example of a false positive with respect to Filter 5.*

Filter 6

In Filter 6 I employ the $xor()$ operation. A bitwise exclusive-OR ($xor()$) operation between two consecutive characters of a string $\mathcal{P}$ of length m is defined as follows: $xor(\mathcal{P}[i], \mathcal{P}[i+1]) = \mathcal{P}_N[i] \wedge \mathcal{P}_N[i+1]$, where $1 \leq i \leq m-1$. I define $sum\_xor(\mathcal{P})$ to be the summation of the results of the xor operations on the consecutive characters of $\mathcal{P}$.

More formally, $sum\_xor(P) = \sum_{i=1}^{m-1} xor(\mathcal{P}[i], \mathcal{P}[i+1])$. Now I present the following observation which is the basis of Filter 6. Note that this observation is applied on the extended versions of the respective strings.

Observation 6 *Consider a circular string $\mathcal{P}$ and a linear string $\mathcal{T}$ both having length n and assume that $\mathcal{A} = ext(\mathcal{P})$ and $\mathcal{B} = ext(\mathcal{T})$. If $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$, then, I must have*

$$sum\_xor(\mathcal{B}) - k \times 14 + k \times 0 \leq sum\_xor(\mathcal{C}(\mathcal{A}))$$

$$\leq sum\_xor(\mathcal{B}) + k \times 14 + k \times 0.$$

Note carefully that the function $sum\_xor()$ has been applied on the extended strings.

Example 7 *Consider the same four strings of Example 3, i.e., $\mathcal{P} = atcgatg$, $\mathcal{T}1 = aacgatg$ and $\mathcal{T}2 = ttcgatg$. As can be easily verified, here $\mathcal{P} \equiv_1 \mathcal{T}1$ and $\mathcal{P} \equiv_1 \mathcal{T}2$. Now consider the extended strings and assume that $\mathcal{A} = ext(\mathcal{P})$, $\mathcal{B}1 = ext(\mathcal{T}1)$ and $\mathcal{B}2 = ext(\mathcal{T}2)$. It can be easily verified that $sum\_xor(\mathcal{C}(\mathcal{A}))$ is 28. Recall that $\mathcal{T}1$ is slightly different from $\mathcal{P}$, i.e, $\mathcal{P}[2] = t \neq \mathcal{T}1[2] = a$. Now I have $\mathcal{T}1_N = 1123143$. Hence $\mathcal{B}1_N = 11231431$. Hence, $sum\_xor(\mathcal{B}1) = 20$ which is indeed within the bounds of Observation 6. Now consider $\mathcal{T}2$, which is slightly different from $\mathcal{P}$, i.e, $\mathcal{P}[1] = a \neq \mathcal{T}2[1] = t$. Now I have $\mathcal{T}2_N = 4423143$. Hence $\mathcal{B}2_N = 44231434$. Hence,$sum\_xor(\mathcal{B}2) = 28$, which is also within the bounds. Finally, consider $\mathcal{T}' = atagctg$, which is again slightly different from $\mathcal{P}$. It can be easily verified that $\mathcal{C}(\mathcal{P}) \not\equiv_1 \mathcal{T}'$. However, assuming that $\mathcal{B}' = ext(\mathcal{T}')$ I find that $sum\_xor(\mathcal{B}')$ is still 28, which is in the range of Observation 6. This is an example of a false positive with respect to Filter 6.*

3.3.3 Reduction of search space in the text

Now I present an $\mathcal{O}(n)$ time algorithm to reduce the search space of the text applying the six filters presented above. It takes as input the pattern $\mathcal{P}[1:m]$ of length m and

Algorithm 3 Approximate Circular Pattern Signature using Observations $1 : 6$ in a single pass

1: **procedure** $ACPS\_FT(\mathcal{P}[1:m])$
2: define three variables for observations 1, 2, 3, 5, 6
3: define an array of size 4 for observation 4
4: define an array of size 4 to keep fixed value of A, C, G, T
5: $s \leftarrow \mathcal{P}[1:m]\mathcal{P}[1]$
6: initialize all defined variables to zero
7: initialize fixed array to $\{1, 2, 3, 4\}$
8: **for** $i \leftarrow 1$ to $|s|$ **do**
9: **if** $i \neq |s|$ **then**
10: calculate different filtering values via observations 1 & 4 and make a running sum
11: **end if**
12: calculate different filtering values via observations 2, 3, 5 & 6 and make a running sum
13: **end for**
14: **return** all observations values
15: **end procedure**

the text $\mathcal{T}[1:n]$ of length n. It calls Procedure $ACPS\_FT$ with $\mathcal{P}[1:m]$ as parameter and uses the output. It then applies the same technique that is applied in Procedure $ACPS\_FT$ (*Algorithm* 3). I apply a sliding window approach with window length of m and calculate the values applying the functions according to Observations $1 : 6$ on the factor of $\mathcal{T}$ captured by the window. Note that for Observations 2, 3, 5 and 6, I need to consider the extended string and hence the factor of $\mathcal{T}$ within the window need be extended accordingly for calculating the values. After calculating the values for a factor of $\mathcal{T}$, I check it against the returned values of Procedure $ACPS\_FT$. If it matches, then I output the factor to a file. Note that in case of overlapping factors (e.g., when the consecutive windows need to output the factors to a file), Procedure $ACPS\_FT$ outputs only the non-overlapped characters. And Procedure $ACPS\_FT$ uses a \$ marker to mark the boundaries of non-consecutive factors, where $\$ \notin \Sigma$.

Now note that I can compute the values of consecutive factors of $\mathcal{T}$ using the sliding window approach quite efficiently as follows. For the first factor, i.e., $\mathcal{T}[1..m]$ I

Algorithm 4 Reduction of Search Space in a Text String using procedure ACPS_FT

procedure $RSS\_FT(\mathcal{T}[1:n], \mathcal{P}[1:m])$
  CALL CPS_FT($\mathcal{P}[1:m]$)
  save the return value of observations $1:6$ for further use here
  define an array of size 4 to keep fixed value of A, C, G, T
  initialize fixed array to $\{1, 2, 3, 4\}$
  $lastIndex \leftarrow 1$
  for $i \leftarrow 1$ to m **do**
    calculate different filtering values in $\mathcal{T}[1:m]$ via observations $1:6$ and make a running sum
  end for
  if $1:6$ observations values of $\mathcal{P}[1:m]$ vs $1:6$ observations values of $\mathcal{T}[1:m]$ have a match **then**
    ▷ Found a filtered match
    Output to file $\mathcal{T}[1:m]$
    $lastIndex \leftarrow m$
  end if
  for $i \leftarrow 1$ to $n - m$ **do**
    calculate different filtering values in $\mathcal{T}[1:m]$ via observations $1:6$ by subtracting i-th value along with wrapped value and adding $i + m$-th value and new wrapped vale to the running sum
    if $1:6$ filtering values of $\mathcal{P}[1:m]$ vs $1:6$ filtering values of $\mathcal{T}[i+1:i+m]$ have a match **then**
      ▷ Found a filtered match
      if $i > lastIndex$ **then**
        Output an end marker $ to file
      end if
      if $i + m > lastIndex$ **then**
        if $i < lastIndex$ **then**
          $j \leftarrow lastIndex + 1$
        else
          $j \leftarrow i + 1$
      end if
      Output to file $\mathcal{T}[j:i+m]$
      lastIndex $\leftarrow i + m$
    end if
  end if
end for
end procedure

exactly follow the strategy of Procedure $ACPS\_FT$. When it is done, I slide the window by one character and I only need to remove the contribution of the left most character of the previous window and add the contribution of the rightmost character of the new window. The functions are such that this can be done very easily using simple constant time operations. The only other issue that needs be taken care of is due to the use of the extended string in two of the filters. But this too does not need more than a few simple constant time operations. Therefore, overall runtime of the algorithm is $\mathcal{O}(m) + \mathcal{O}(n - m) = \mathcal{O}(n)$. The algorithm is presented in the form of Procedure RSS_FT (Algorithm 4).

3.3.4 The combined algorithm

I have already described the two main components of the algorithm, namely, Procedure $ACPS\_FT$ and Procedure RSS_FT, which in fact calls the former. Now Procedure RSS_FT provides a reduced text $\mathcal{T}'$ (say) after filtering. At this point I can use any algorithm that can solve ACPM and apply it over $\mathcal{T}'$ and output the occurrences. Now, suppose I use Algorithm $\mathcal{A}$ at this stage which runs in $\mathcal{O}(f(|\mathcal{T}'|))$ time. Then, clearly, the overall running time of the approach is $\mathcal{O}(n)+\mathcal{O}(f(|\mathcal{T}'|))$. In the implementation I have used the recent algorithm of [4]. In particular, in [4], the authors have presented an approximate circular string matching algorithm with k-mismatches (ACSMF-Simple) via filtering. They have built a library for ACSMF-Simple algorithm. The library is freely available and can be found here: [13]. I only apply ACSMF-Simple on the reduced string.

3.4 Experimental Results

I have implemented SimpLiFiACPM and conducted extensive experiments to analyze its performance. I show the comparison based on the experimental result between ACSMF-Simple [4] and the algorithm SimpLiFiACPM. ACSMF-Simple [4] has been

implemented as library functions in the *C* programming language under *GNU/Linux* operating system. The library implementation is distributed under the GNU General Public License (GPL). It takes as input the pattern $\mathcal{P}$ of length m, the text $\mathcal{T}$ of length n, and the integer threshold $k < m$ and returns the list of starting positions of the occurrences of the rotations of $\mathcal{P}$ in $\mathcal{T}$ with k-mismatches as output.

3.4.1 Dataset

I have used real genome data in the experiments as the text string, $\mathcal{T}$. This data has been collected from [62]. Here, we have taken *1GB* of data for the experiments. I have generated random patterns of different length by a random indexing technique in this *1GB* of text string.

3.4.2 Environment

I have conducted the experiments on a PowerEdge R820 rack server PC with 6-core of Intel Xeon processor *E5-4600* product family and 64GB of RAM under GNU/Linux. I have coded SimpLiFiACPM in C++ using a GNU compiler with General Public License (GPL). As it has been mentioned already above, the implementation of SimpLiFiACPM uses the ACSMF-Simple [4]. With the help of the library used in [4], I have compared the running time of ACSMF-Simple of [4] and SimpLiFiACPM.

3.4.3 Experiments

Because I have 6 different filters, the order of applying the filter may turn out to be important in the final performance of the algorithm. I have done preliminary experiments to identify the best order to apply the filters. I have run this experiment on three candidate patterns of different length. As there could be 6!, i.e., 720 combinations, I have recorded the running times of the algorithm for all these combinations considering the three candidate patterns. Although it was not possible to identify one

TABLE 3.1: Elapsed-time (in seconds) and speed-up comparisons among ACSMF-Simple[4] and the algorithm considering all the six filters for a text of size 1GB

m	k	Elapsed Time(s) of ACSMF-simple	Elapsed Time(s) of Simplify ACPM	Speed up of Simplify ACPM
2000	2	32.827	16.345	2
3000	2	28.464	13.650	2
4000	2	32.286	15.456	2
5000	5	39.577	18.456	2
6000	5	35.848	15.345	2
2000	5	38.333	16.986	2
3000	5	39.900	18.976	2
4000	5	28.339	12.345	2
5000	5	39.577	20.436	2
6000	5	35.848	14.547	2
5000	7	44.173	19.546	2
6000	7	64.362	28.435	2
7000	7	59.190	24.879	2
8000	7	63.095	32.435	2
9000	7	58.588	26.789	2
10000	10	45.601	20.456	2
50000	10	46.683	20.436	2
100000	10	100.306	45.677	2
150000	10	105.188	55.567	2
200000	10	122.252	62.345	2
250000	12	125.583	62.988	2
300000	12	137.835	71.345	2
350000	12	145.008	75.345	2
400000	12	155.451	72.345	2
450000	12	170.058	76.345	2
500000	12	200.165	80.984	2
550000	15	250.813	100.436	2
650000	15	270.142	120.454	2
700000	15	292.497	140.546	2
750000	15	330.475	172.345	2
800000	15	401.183	193.349	2
850000	15	498.335	250.567	2
900000	20	560.817	240.568	2
950000	20	705.784	300.567	2
1000000	20	870.142	370.546	2
1100000	20	901.033	450.891	2
1150000	20	930.599	530.567	2
1200000	20	955.113	552.345	2

TABLE 3.2: Elapsed-time (in seconds) and speed-up comparisons among ACSMF-Simple[4] and SimpLiFiACPM-[1..3] (considering first three combination of the filters) for a text of size 1GB

m	k	Elapsed Time(s) of ACSMF-simple	Elapsed Time(s) of Simplify ACPM	Speed up of Simplify ACPM
2000	2	32.827	26.488	1
3000	2	28.464	26.333	1
4000	2	32.286	25.856	1
5000	5	39.577	24.185	2
6000	5	35.848	24.021	1
2000	5	38.333	24.015	2
3000	5	39.900	23.983	2
4000	5	28.339	23.837	1
5000	5	39.577	24.185	2
6000	5	35.848	24.021	1
5000	7	44.173	30.282	1
6000	7	64.362	26.983	2
7000	7	59.190	27.087	2
8000	7	63.095	27.333	2
9000	7	58.588	26.615	2
10000	10	45.601	26.050	2
50000	10	46.683	70.993	1
100000	10	100.306	57.942	2
150000	10	105.188	758.018	0
200000	10	122.252	536.863	0
250000	12	125.583	453.232	0
300000	12	137.835	101.059	1
350000	12	145.008	272.283	1
400000	12	155.451	334.928	0
450000	12	170.058	449.691	0
500000	12	200.165	311.247	1
550000	15	250.813	506.227	0
650000	15	270.142	646.575	0
700000	15	292.497	1126.407	0
750000	15	330.475	1213.848	0
800000	15	401.183	1349.880	0
850000	15	498.335	1579.911	0
900000	20	560.817	1289.475	0
950000	20	705.784	894.105	1
1000000	20	870.142	259.141	3
1100000	20	901.033	1328.620	1
1150000	20	930.599	1636.091	1
1200000	20	955.113	1203.015	1

ordering that performs best in all instances, but it appeared that applying filter 4 first produces best result in all instances.

3.4.4 Experimental Results

Here I represent the main experimental results and comparisons between my algorithm and ACSMF-Simple of [4]. Table 3.1 reports the elapsed time and speed-up comparisons for various pattern sizes $(2000 \leq m \leq 1200000)$ and for various mismatch sizes $(2 \leq k \leq 20)$. As can be seen from Table 3.1, the algorithm runs faster than ACSMF-Simple in all cases.

In order to analyze and understand the effect of the filters I have run a second set of experiments as follows. I have run experiments on three variants of SimpLiFiACPM where the first variant (SimpLiFiACPM-[1..3]) only employs Filters 1 through 3, the second variant (SimpLiFiACPM-[1..4]) only employs Filters 1 through 4, and finally the third variant (SimpLiFiACPM-[1..5]) employs Filters 1 through 5. Table 3.2, 3.3 and 3.4 reports the elapsed time and speed-up comparisons considering various pattern sizes $(2000 \leq m \leq 1200000)$ for ACSMF-Simple and the above-mentioned three variants of SimpLiFiACPM. As can be seen from Table 3.2, 3.3 and 3.4, ACSMF-Simple is able to beat SimpLiFiACPM-[1..3] in a number of cases. However, SimpLiFiACPM-[1..4] and SimpLiFiACPM-[1..5] significantly run faster than ACSMF-Simple of [4] in all cases. This indicates that as more and more effective filters are imposed, the algorithm performs better. The program source code link is available at 9.

TABLE 3.3: Elapsed-time (in seconds) and speed-up comparisons among ACSMF-Simple[4] and SimpLiFiACPM-[1..4] (considering first four combination of the filters) for a text of size 1GB

m	k	Elapsed Time(s) of ACSMF-simple	Elapsed Time(s) of Simplify ACPM	Speed up of Simplify ACPM
2000	2	32.827	16.544	2
3000	2	28.464	15.733	2
4000	2	32.286	12.564	3
5000	5	39.577	14.456	3
6000	5	35.848	14.220	3
2000	5	38.333	10.167	4
3000	5	39.900	13.976	3
4000	5	28.339	11.456	2
5000	5	39.577	12.458	3
6000	5	35.848	14.434	2
5000	7	44.173	22.439	2
6000	7	64.362	34.566	2
7000	7	59.190	27.546	2
8000	7	63.095	23.335	3
9000	7	58.588	25.547	2
10000	10	45.601	26.568	2
50000	10	46.683	44.456	1
100000	10	100.306	50.466	2
150000	10	105.188	70.436	1
200000	10	122.252	100.345	1
250000	12	125.583	102.353	1
300000	12	137.835	90.346	2
350000	12	145.008	101.345	1
400000	12	155.451	100.435	2
450000	12	170.058	120.435	1
500000	12	200.165	99.435	2
550000	15	250.813	100.435	2
650000	15	270.142	130.435	2
700000	15	292.497	150.455	2
750000	15	330.475	220.435	1
800000	15	401.183	205.456	2
850000	15	498.335	350.435	1
900000	20	560.817	330.435	2
950000	20	705.784	450.455	2
1000000	20	870.142	430.345	2
1100000	20	901.033	500.546	2
1150000	20	930.599	600.435	2
1200000	20	955.113	700.345	1

TABLE 3.4: Elapsed-time (in seconds) and speed-up comparisons among ACSMF-Simple[4] and SimpLiFiACPM-[1..5] (considering first five combination of the filters) for a text of size 1GB

m	k	Elapsed Time(s) of ACSMF-simple	Elapsed Time(s) of Simplify ACPM	Speed up of Simplify ACPM
2000	2	32.827	15.435	2
3000	2	28.464	17.434	2
4000	2	32.286	13.435	2
5000	5	39.577	20.234	2
6000	5	35.848	17.435	2
2000	5	38.333	15.345	2
3000	5	39.900	19.235	2
4000	5	28.339	14.455	2
5000	5	39.577	17.343	2
6000	5	35.848	20.234	2
5000	7	44.173	20.324	2
6000	7	64.362	30.435	2
7000	7	59.190	25.346	2
8000	7	63.095	20.344	3
9000	7	58.588	30.235	2
10000	10	45.601	25.456	2
50000	10	46.683	45.465	1
100000	10	100.306	52.343	2
150000	10	105.188	69.324	2
200000	10	122.252	78.346	2
250000	12	125.583	90.468	1
300000	12	137.835	93.435	1
350000	12	145.008	140.435	1
400000	12	155.451	102.345	2
450000	12	170.058	72.345	2
500000	12	200.165	101.234	2
550000	15	250.813	120.435	2
650000	15	270.142	140.325	2
700000	15	292.497	145.345	2
750000	15	330.475	170.435	2
800000	15	401.183	207.345	2
850000	15	498.335	330.985	2
900000	20	560.817	340.786	2
950000	20	705.784	432.546	2
1000000	20	870.142	542.435	2
1100000	20	901.033	600.345	2
1150000	20	930.599	605.234	2
1200000	20	955.113	650.234	1

Chapter 4

Client Side Web tool For Circular Pattern Matching

4.1 Web tool

This chapter deals with a client side development approach for Circular Pattern Matching (CPM) problem. I explained in previous chapter in great detail about CPM which appears as an interesting problem in many biological contexts. In short, CPM is to find all possible occurrences of the rotations of a pattern $\mathcal{P}$ of length m in a text $\mathcal{T}$ of length n with k mismatches. I have developed a lightweight and fast web based tool for users. Much of the speed can be attributed to the fact that actual computation is done in client machine instead of uploading big chunks of data in the server. I have uploaded the web tool on http://samirsoftware.com/acpm/index.html [63]. The source codes are available in my repository http://www.samirsoftware.com/Tool.zip [64]. The source code link is available at 9 too.

4.2 Contribution

The main contribution in this chapter is a fast and efficient browser based tool which dramatically reduces the size of data by using filter based approximate circular pattern matching approach and find out the match. To my knowledge while this thesis

is underway, this is the first web based circular pattern matching tool that operates in client side. The idea behind the approach is quite simple and intuitive. The users do not need to install any software or do not need to upload the big file in the server. My development approach works with big data in client side by using lower memory working as chunk by chunk without uploading any data to web server. Instead of program running in the web server, the java script code is transported in the browser in run-time and does the computation. The user does not require to install any software because the computation is done in the browser. I presented a web based tool for an effective lightweight filtering technique to reduce the search space of the Circular Pattern Matching problem which works for both exact and approximate circular pattern matching. I worked on 3 GB data but it can easily work with large data because the memory does not load the data at a time. It works chunk by chunk.

4.3 Description of the tool

The Approximate Circular String Search online tool searches an input text for matches of an input pattern. A match can either be the original input pattern or any rotation of the input pattern. A match can be defined by the user to be exact or approximate within the user specified bound k.

The text is first filtered by 3 filters to a reduced search space, represented by a list of 0-indexed (index starts from 0) locations in the text. These indices represent the location of candidate matches and may include false positives. However these false positives can optionally be removed through an additional verification process to produce a final definitive list of matches.

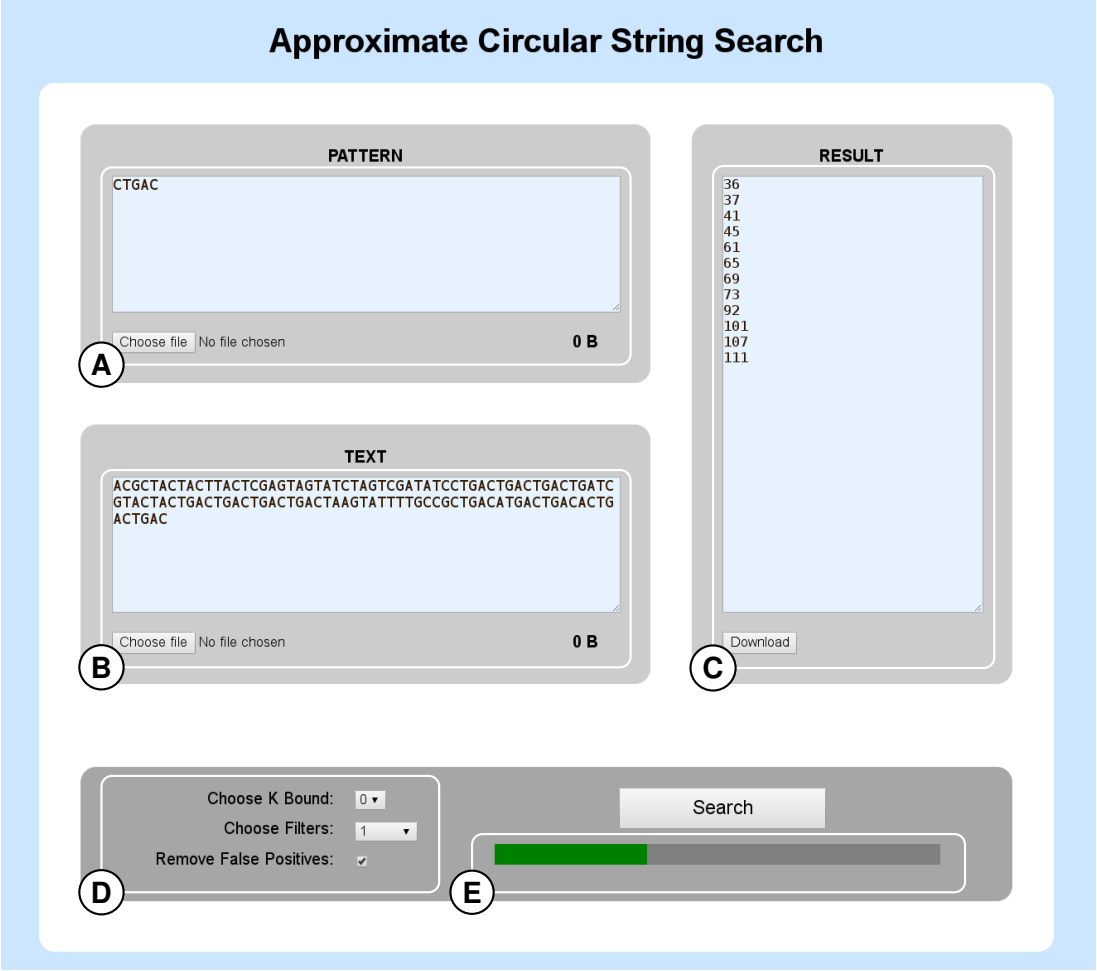


FIGURE 4.1: Web tool interface

A, B Pattern and Text Input: Regions allow a user to enter data to be used in the string search. These can be entered manually or added from a text file. Data in these files is not uploaded to the web server, instead the algorithm will run on the client-side where the data is located.

C Results Window: Displays the 0-indexed locations in the text where potential or precise matches are found. A user may download this data to a file for further analysis.

D Algorithm Options: User specifies the parameters by which the search algorithm should operate.

- *K Bound:* Describes the maximum number of mismatches between pattern and text that may be allowed. K = 0 is equivalent to exact circular string matching. K = 1 means all but 1 character was matched etc.
- *Filters:* There are 3 filters that may be applied in the filtering stage. Applying additional filters may increase the search time but reduces the size of the final list of candidate locations. I have done 720 combinations of experiments based on 6 filters to understand which order of filters produce the better result. Clearly filter 4 is the winner that performs better when it is in the first position in the experiment in terms of execution time for each variation of input size. But it is not possible to come into conclusion on the serial of other filters conclusively. However, by manual observation of the filters, it appears the following filters perform better. So I have chosen the following 3 filters and reiterating the observation below. All usual notations below convey the same meaning as of Chapter 3. In my tool, I am calling the following three observation as Filter 1, 2 and 3 in combo box of chosen filters for this tool.

  1. **Observation 1** *Consider a circular string $\mathcal{P}$ and a linear string $\mathcal{T}$ both having length n. If $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$, where $0 \leq k < n$, then I must have*

  $$sum(\mathcal{T}) - k \times 4 + k \times 1 \leq sum(\mathcal{C}(\mathcal{P})) \leq sum(\mathcal{T}) + k \times 4 - k \times 1.$$

  2. **Observation 2** *Consider a circular string $\mathcal{P}$ and a linear string $\mathcal{T}$ both having length n and assume that $\mathcal{A} = ext(\mathcal{P})$ and $\mathcal{B} = ext(\mathcal{T})$. If $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$, where $0 \leq k < n$, then I must have*

  $$abs\_total\_distance(\mathcal{B}) - k \times 4 + k \times 1 \leq$$

$$abs\_total\_distance(\mathcal{C}(\mathcal{A})) \leq$$

$$abs\_total\_distance(\mathcal{B}) + k \times 4 - k \times 1.$$

3. **Observation 3** *Consider a circular string $\mathcal{P}$ and a linear string $\mathcal{T}$ both having length n. If $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$, where $0 \leq k < n$, then I must have*

$$sum_x(\mathcal{T}) - k \times num(x) \leq sum_x(\mathcal{C}(\mathcal{P}))$$

$$\leq sum_x(\mathcal{T}) + k \times num(x)$$

for all $x \in \Sigma$.

- *Remove False Positives:* If this option is not checked, the list of locations produced will represent potential locations of matches and may contain false positives. If this option is checked, the list of locations will be the final correct list of match locations.

E Progress Bar: Monitors progress of algorithm execution.

Chapter 5

Web tools security in development perspective

5.1 Problem Definition of Web Vulnerabilities

Web tools attracts the inherent vulnerabilities of web. The common web vulnerabilities to date from software security research illustrated by the CWE/SANS on top 25 most dangerous software errors report [47] in which web application vulnerabilities ranked the top. In addition, the industry scanning tools[48] gives the insight about the issues. According to the Open Web Application Security Project (OWASP) the most common web vulnerabilities are SQL Injection, Cross-site Scripting (XSS), Cross-Site Request Forgery (CSRF), Security Misconfiguration, Unvalidated Redirects and Forwards, No Account Lockout, Insecure Password Policy and HTTPS not enforced as presented in their Top Ten Most Critical Web Application Security Risks Report [49]. In most cases the hackers use these common vulnerabilities to get access to the system which later turns out to be disastrous for the organizations [47].

5.1.1 Applications and motivations

While developing web tools for circular string, I came across the web security issue. Most sites in current web spectrum are vulnerable. In fact, 86% of websites contain at

least one 'serious' vulnerability according to the WhiteHat Security Statistics Report 2015 [46]. In this chapter, the vulnerabilities mentioned here has been addressed in a structured algorithmic manner. Providing very easy and effective guidelines and solutions in practical context is the main motivations of this chapter. The program source code link is available at 9.

5.1.2 Organization of the chapter

The rest of the chapter is organized as follows. The three most serious vulnerabilities (SQL injection, Cross-site Scripting (XSS), Cross-Site Request Forgery (CSRF)) are treated in separate section **FixWebSQLInjection** on 5.2, **FixWebXSS** 5.3, **FixWebCSRF** 5.4 respectively. The rest of the vulnerabilities are treated together in section **FixOtherVulnerabilities**5.5.

5.1.3 My contribution

This chapter presents guidelines and solutions for each vulnerability. The proposed solutions are described in separate sections in an algorithmic manner. The solutions came across by learning the reasons of vulnerability[47] and penetration technique used by the scanning tools [48] to exploit the vulnerability. The vulnerability details have been described briefly to cover the basics of common vulnerabilities. I have given detail description of the solutions in the next few sections. A very brief overview of each problem and the solutions with the best practices as well as PSEUDO-CODE has been provided in the chapter. The relevant example codes with vbscript and javascript has been uploaded in the url [65]. The javascript code I have provided can be plugged directly in web application at the client side because javascript is de facto standard of browser based client side language.

5.2 FixWebSQLInjection

5.2.1 Vulnerability detail

As mentioned previously, SQL Injection was stated in top of the list in the CWE/SANS Top 25 Most Dangerous Software Errors report [47]. XSS attacks is in rank 4 and the CSRF attacks in number 12. That is a strong proof of how these three vulnerabilities are growing rapidly throughout the year among the other web and software security issues.

SQL is a Structured Query Language that manage (add, delete, edit, and retrieve) the stored data in a relational database. It is the most popular way for users to communicate with the database. Furthermore, SQL injection attack is an attack in which the malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution.

In other words, SQL Injection attack occurs when untrusted data is sent to an interpreter as part of a command or query. In web perspective that can be done by the attacker provides SQL code to a user input box through the form in the web application to gain unauthorized access to the database. Any procedure that constructs SQL statements should be reviewed for injection vulnerabilities because SQL Server will execute all syntactically valid queries that it receives. Even parametrized data can be manipulated by a skilled and determined attacker [66].

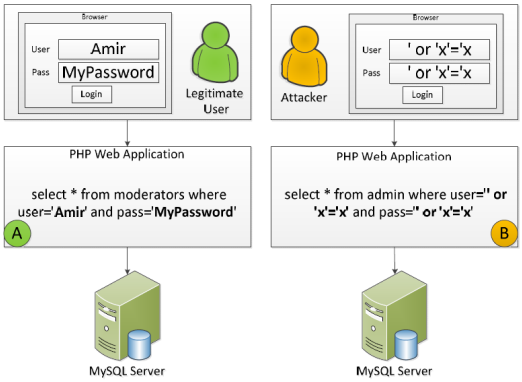


FIGURE 5.1: SQL Injection attack using a login form

[67]

SQL injection is a threat for any data driven applications regardless of their underlying databases. It is considered to be the most effective method for stealing data from the back-end nowadays. However, there are many forms of SQL injection either based on the method of injection (through user input, through cookies, through server variables or second-order injection) or based on the attacker intention [68] with the injectable parameters such as performing database finger-printing, determining database schema, extracting data, adding or modifying data, performing denial of service, evading detection, bypassing authentication, executing remote commands or performing privilege escalation etc. However, the technical form of SQL Injection can be categorized according to the type of the attack against the databases.

- ***SQL Manipulation:*** By modifying the SQL statement in different ways for example, through using different operations such as UNION or by changing the WHERE clause of the statement to retrieve different outputs or through some error messages that contain the table and column name which will help the attacker to extract the required data.

- ***Code Injection:*** by injecting malicious code when inserting new SQL statement, for example, after extracting the table and column's name by SQL manipulation

method, these can be inserted in SQL statement to extract information to retrieve more data from database.

- ***Function Call Injection:*** The attacker can execute built in procedure through calling functions to manipulate the data in the database .
- ***Buffer Overflow:*** By providing more data to an input variable more than the amount of space allocated for that specific input, causing overwriting the memory elsewhere in the application to modify it.

Interested readers are referred to read more on [68].

5.2.2 Solutions

This solution here is to provide with the generic approach. It divides the solutions of SQL injection to three categories by which malicious query can be passed through database via web applications.

5.2.3 Type check

It needs to trap each query-string and form fields based on what application is expecting. Input length, boundaries and expected values must have to be trapped. If an input field passes a value that does not fit with the requirements of that field the specific web application is expecting, must have to be trapped.

The solution deals with Boolean Algorithm 5, Byte Algorithm 6, Currency Algorithm 7, Date Algorithm 8, Double Algorithm 9, Integer Algorithm 10 and String Algorithm 11. These few data types are just for guidance purpose. Users must have to trap all different data type explicitly that the query is expecting from each individual form field in the individual web application. If any of the data type doesn't match

with the expected input for an individual form field, then it has to be considered as a breach. This kind of error sometimes are genuine mistake by user and sometimes could be hacking attempt. For example when user mistakenly types phone number in the address field, the error will be trapped. So the error must have to be managed with proper error message so that user can get help. In the same time, the hackers cannot bypass with the wrong data type.

Algorithm 5 IsValidBoolean(S)

```
1: procedure ISVALIDBOOLEAN(S)
2:     if S is boolean then
3:         return TRUE
4:     else
5:         return ERROR with possible breach
6:     end if
7: end procedure
```

Algorithm 6 IsValidByte(S)

```
1: procedure ISVALIDBYTE(S)
2:     TEST programmatically whethere S is a BYTE
3:     if S is BYTE then
4:         return TRUE
5:     else
6:         return RETURN ERROR with possible breach
7:     end if
8: end procedure
```

Algorithm 7 IsValidCurrency(S)

```
1: procedure ISVALIDCURRENCY(S)
2:     TEST programmatically whethere S is a Currency
3:     if S is a Currency then
4:         return TRUE
5:     else
6:         return RETURN ERROR with possible breach
7:     end if
8: end procedure
```

Algorithm 8 IsValidDate(S)

```
1: procedure ISVALIDDATE(S)
2:     TEST programmatically whethere S is a date
3:     if S is a Date then
4:         return TRUE
5:     else
6:         return RETURN ERROR with possible breach
7:     end if
8: end procedure
```

Algorithm 9 IsValidDouble(S)

```
1: procedure ISVALIDDOUBLE(S)
2:     TEST programmatically whethere S is a double
3:     if S is a Double then
4:         return TRUE
5:     else
6:         return RETURN ERROR with possible breach
7:     end if
8: end procedure
```

Algorithm 10 IsValidInteger(S)

```
1: procedure ISVALIDINTEGER(S)
2:     TEST programmatically whethere S is an integer
3:     if S is a Integer then
4:         return TRUE
5:     else
6:         return RETURN ERROR with possible breach
7:     end if
8: end procedure
```

Algorithm 11 IsValidString(S)

```
1: procedure ISVALIDSTRING(S)
2:     TEST programmatically whethere S is a string
3:     if S is a string then
4:         return TRUE
5:     else
6:         return RETURN ERROR with possible breach
7:     end if
8: end procedure
```

5.2.4 Length, minimum, maximum and regular expression check

Expected length of each form field based on database field's length must have to be trapped. The minimum and maximum value of the field need to be trapped in both client and server side logic. From the usability perspective the minimum and maximum boundary needs to be defined in application design so that each field can be trapped. We have written here two version of such trap. The former is just plain minimum and maximum check and the later one is with the regular expression. Regular expression can dictate what is allowed or not in a specific filed.

For example, Name field can not have numbers or special characters at all, phone number cannot have any letters or special characters etc. It is possible to trap by writing the regular expression and validating the input with the routine before passing this into query. Based on the practitioner's need for a specific form field, any of the procedures Validate Min Max 12 or Validate Min Max with Regular Expression 13 can be used.

Algorithm 12 IsValidMinMaxn(S, intMinLength, intMaxLength)

1: **procedure** $IsValidMinMaxn$(String S, Integer intMinLength, Integer intMaxLength)
2: **if** $0 = intMaxLength$ **then**
3: $intMaxLength$ = $Len(varTemp)$
4: **end if**
5: **if** $LENGTH(S) < intMinLength$ Or $LENGTH(S) > intMaxLength$ **then**
6: EXIT $IsValidMinMax$ with ERROR
7: **end if**
8: **end procedure**

Algorithm 13 IsValidTexWithREgularExpression(S, iMinLen, iMaxLen , regexp)

1: **procedure** $IsValidTexWithREgularExpression$(S, iMinLen, iMaxLen , regexp)
2: **if** (LENGTH(S)>= iMinLen && LENGTH(S) <= iMaxLen) && (!LENGTH(S) || regexp.test(S)) **then**
3: RETURN TRUE
4: **else**
5: EXIT IsValidMinMax with ERROR
6: **end if**
7: **end procedure**

5.2.5 SQL safe query

In SQL injection there are common words, characters and syntax those are used for attack. Developers can write a common functionality to address this issue. We have written a common routine SQLSafe 14. Over time the practitioners can modify by adding new vulnerable characters trap in the routine those possibly are used in the SQL injection.

Algorithm 14 SQLSafe(S)

1: **procedure** $SQLSafe(S)$
2: DECLARE cleanstr
3: **if** $LENGTH(S) = 0$ **then**
4: EXIT SQLSafe
5: **end if**
6: **if** "′)OR′"Exist in S **then**
7: EXIT SQLSafe with ERROR
8: **end if**
9: **if** "′)AND′"$ExistinS$ **then**
10: EXIT SQLSafe with ERROR
11: **end if**
12: REPLACE "’" WITH ""
13: REPLACE "–" Or "-" WITH ""
14: REPLACE `xp_cmdshell`, `xp_log70.dll`, openrowset, `sp_makewebtask` WTIH "`d99_temp`"
15: TRIM S
16: **return** S
17: **end procedure**

5.2.6 Use a middle tier

It is a best practice to use a middle tier or at least a database layer to have a greater flexibility in managing the vulnerability. This will help trapping the SQL injection in application level directly before the database call. Using stored procedure or separate function for query is a best practice of such modular implementation. Our procedure StoredProcedureCall 15 gives the guidelines how to call a stored procedure.

Before calling the stored procedure or separate data driven function, the practitioner will need to validate the parameters datatype, each SQL field's data length and the exact number of parameters explicitly assigned. Any violation on length, data type and number of parameters nwill result in a failure and will be considered as a possible security breach.

The same technique can be used for desktop based application too, although the focus of our write-up us web application. SQL injection can happen via XSS attack too which has been described in FixWebXSS section 5.3. So the solution on procedure CrossScriptingTest 16 will need to be called from server side too before calling the databse. In the codes [65] two versions have been written. CrossScriptingTest has been written in classic asp vbscript code[65] for server side implementation and CrossScriptingTestJS[65] has been written in javascript for using from client side.

Algorithm 15 StoredProcedureCall

```
1: procedure STOREDPROCEDURECALL
2:     Explicitly check each parameters datatype
3:     Explicitly check each parameters datalenth defined in SQL
4:     if both of above matches then
5:         make the SQL call
6:     else
7:         return ERROR
8:     end if
9: end procedure
```

5.3 FixWebXSS

5.3.1 Vulnerability detail

Cross-site scripting (XSS) is one of the most prevalent, obstinate, and dangerous vulnerabilities in web applications[47]. Yet, it is still remains as a big problem for web applications, despite the bulk of solutions provided so far[69]. The XSS attack occurs by injecting a malicious Javascript or other untrusted browser-executable content into a web page that the user application generates. And that is considered as a result of a failure to validate the inputs from the web site users.

In other words, XSS attacks is a result of weak input validation on the web application that will permit the attacker to execute the malicious scripts that was injected on the victim's web browser to steal data, cookies, passwords etc or the code may be sent to another user and executed on their browser cause serious security exploitations.[1][70]

Although the XSS is an attack on the client-side web browser, but it will gain the attacker an access to the web server side as well.

5.3.2 XSS injection process

There are different techniques to inject the malicious Java Script code into the victim's web application. However, the easiest way is through the web application that accepts inputs from the user side and that is usually the case in dynamic web applications or through utilizing the plug-ins for example (Flash XSS vulnerability). Furthermore, the XSS attacks requires three parties (attacker web application, victim web application and victim web browser).

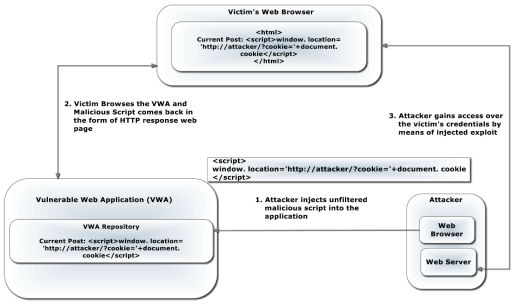


FIGURE 5.2: Cross-Site Scripting exploitation [1]

Cookie stealing: Cookies are the most sensitive Information that a Web site saves about each user However, XSS attacker can steal them by injecting the XSS attack vectors on the vulnerable website for intention of stealing these cookies specially the session IDs or as it called (session hijacking).

Phishing attacks: The XSS attacker can steal the victim's credentials information directly from them by injecting the code of a fake log in page identical the original one and trick the victims into entering their log in details in it.

Key logging: The attacker can utilize the capabilities of keyboard event listener to trace all the keystrokes of the victims and transfer this information to the web server for accessing the sensitive information like password, credit card numbers etc.[1]

5.3.3 XSS attacks

***Stored or Persistent XSS attack*:** the attacker will target a web application with XSS vulnerability and inject malicious code and store it on the web application server that will cause damages to the web application or compromising the users data who access it usually the main target of this attack are blogs, forums or profiles.

***Reflected or Non-persistent XSS attack*:** the attacker in this case wont store the malicious code on the web application server but instead will interact with the web application users by either sending malicious links to the victims using email, or embedding the link in a web page residing on another server so after clicking on the malicious link the attack will be sent to the victim's browser and the harm will be effected.

***DOM-based XSS attack*:** in this attack the effected part is the DOM environment which the attacker will modify it to preform unwanted/unexpected actions at the client side while the page is not modified [70]

XSS attacks are still difficult to address. There are ways to minimize the XSS attacks. By encoding and increasing the inputs validation level, it is possible to minimize the attacks [70].

Nevertheless, many approaches to prevent and detect XSS attacks has been proposed with different techniques either Implemented on client side or server side or hybrid technique which are done in combination of client and server side [71]. Interested readers are referred to [72] and [69] for further detail.

5.3.4 Solutions

The solution deals with Cross-site scripting (XSS) attack by two procedures to make sure that the attack is tackled in both client and server side. The procedure OnFormSubmissionInClient 17 will need to be called during form submission which essentially calls procedure CrossScriptingTest 16. Vulnerable syntax and SQL characters

are trapped here to avoid SQL injection via XSS attack.

Algorithm 16 CrossScriptingTest()

```
1: procedure CROSSSCRIPTINGTEST()
2:     DECLARE FORMSDATA
3:     READ All Forms Data into FORMSDATA
4:     REPLACE "%20" with " " in FORMSDATA
5:     REPLACE "%27" with "'" in FORMSDATA
6:     REPLACE "%29" with ")" in FORMSDATA
7: COMMENT: The Replacement list above can grow based on new vulnerability or
   missing syntax
8:     DECLARE HACKINGDATA
9:     ASSIGN the list of possible hacking values in HACKINGDATA
10:    HACKINGDATA[] ={"script","sysdatabase","sysobject", "char(", "varchar", "se-
   lect", "alert", "\x00","\x20", "') or '", "') and '" }
11:    for i=0 to LENGTH_ARRAY (FORMSDATA[]) −1 do
12:        for j=0 to LENGTH_ARRAY(HACKINGDATA[])−1 do
13:            if HACKINGDATA[j] EXISTS in FORMSDATA[i] then
14:                return with possible Hacking attempt
15:            end if
16:        end for
17:    end for
18: end procedure
```

Algorithm 17 OnFormSubmissionInClient

```
1:  procedure ONFORMSUBMISSIONINCLIENT
2:      Call CrossScriptingTest
3:      if PROCEDURE CrossScriptingTest RETURNS 'possible hacking attempt'
    then
4:          RETURN ERROR with possible breach
5:      end if
6:      Call each data_type trap PROCEDURE
7:      if EACH data_type trap RETURNS False then
8:          RETURN ERROR with possible breach
9:      end if
10:     IF NO ERRORS
11:     SUBMIT FORM
12: end procedure
```

5.4 FixWebCSRF

Cross-Site Request Forgery (CSRF) attacks occurs when the victim holds an active session with a trusted site while visiting a malicious site. It creates a malicious request by the attacker to force the victim's web browser to perform an unwanted actions on the trusted website without the victim's interaction in that request by tricking the browser to believe that these requests are actually a legitimate requests from the victim.

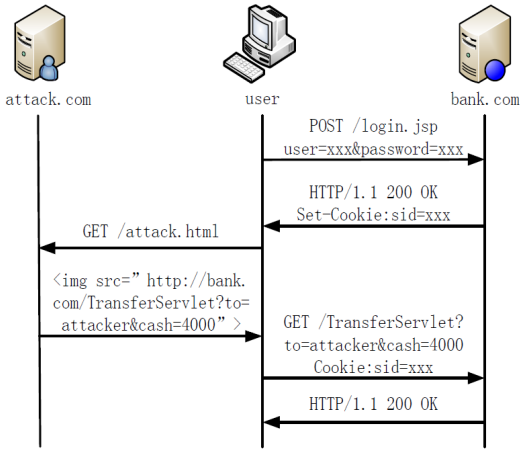


FIGURE 5.3: CSRF attack [2]

XSS and CSRF are two different different form of attack. The cross-site scripting XSS exploits the trust of a user, while the cross-site request forgery exploits the trust of an web application.

There are two types of CSRF attacks (reflected and stored) [2].There are some limitations to the CSRF attack. In order for the CSRF to succeed it depends on some important factors, such as:

- Attacker should analyse the structure of the website and the forms included.
- Finding website that does not check referrer header is an important step in the attack.
- The user should be in a valid session (logged in) in the web site while clicking on the malicious link.

These factors should be true while attacking [73].

5.4.1 Solutions

Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet [74] has described the possible solutions in greater detail. CSRF attacks can be resolved by disabling the previous points such as checking the referrer header and limiting the lifetime of authentication cookies. However limiting the lifetime of the authentication cookies does not help the most data driven site because users require to login to the system more frequently and that is considered to be a distracting factors for the users and business. Interested readers are referred to [73] and [75] for more detail. In practice checking the http Referrer and using secure cookies prevents CSRF attacks greatly.

5.4.2 Check referrer

Maintaning a white list of IP addresses or URL those can send POST data to web applications is important. In most business cases it is possible to restrict post operations from any other server except the actual Web URL. IsValidReferrer 18 can deal with Referrer check based on a white list of IP address or URL. According to [74], Referrer trap should be able to handle most CSRF attacks considering those attack are done by manipulating form and querystring from another server. So clearly the white listing will prevent those server by referrer.

Algorithm 18 IsValidReferrer

```
1: procedure IsValidReferrer
2:     if REQUESTED REFERRER MATCHES WITH A WHITE LIST OF IP ADDRESS OR URL then
3:         RETURN TRUE
4:     else
5:         EXIT IsValidReferrer with POSSIBLE CSRF Breach
6: end if
end procedure
```

5.4.3 Use secure cooking

It is important to use secure cookies. This can be done both is webserver configuration level or from application logic. I suggest of doing both. If any of these are compromised, the other can be operational. Uers are referred to https://msdn.microsoft.com/en-us/library/ms228262(v=VS.80).aspx [76] for setting up secure coookies in server level.

Algorithm 19 SetValidSecureCookies

```
1: procedure SetValidSecureCookies
2:     Check whether the cookies is HTTP or HTTPS
3:     if NOT HTTPS then
4:         Set HTTPS
5:     end if
6: end procedure
```

5.5 FixOtherVulnerabilities

This section deals with Security Misconfiguration, Unvalidated Redirects and Forwards, No Account Lockout, Insecure Password Policy, HTTPS not enforced.

5.5.1 Security misconfiguration

[49] has got a great detail why the site becomes vulnerable due to misconfiguration. Unpatched security flaws in the server software permits directory listing and directory traversal attacks. Unnecessary default backup, applications, configuration files, web pages, improper file and directory permissions,services enabled, including content management and remote administration can make the system vulnerable to attack. Default accounts with their default passwords, administrative or debugging functions that are enabled or accessible, overly informative error messages (more details in the

error handling section), misconfigured SSL certificates and encryption setting, use of default certificates, improper authentication with external systems can cause issue.

Monitoring the operating systems, software and latest security update are essential in the web server which eventually will give the security of the web application on that server.

5.5.2 Unvalidated redirects and forwards

Frequent Redirects and Forwards can be dangerous. Specially the parametrized query-string can be manipulated by hackers by changing the values. [49] suggested to avoid redirect and forward. But in real life web solution, it is difficult to avoid query-string in redirect and forward. So the best solution for redirection is not to trust the query-string and validate each of them by using validation technique described in previous section.

The SQL injection routine and XSS routine described in 5.2 and 5.3 will be useful. The parametrized query-string issue can be resolved by dynamic URL rewriting in server side too. That will disguise the actual query-string and will be hard for the hackers to guess.

5.5.3 No account lockout

Account must have to be locked out after several tries. Otherwise it will lead to Brute Force Attack[49]. Although CAPTCHA is considered to be a solution but this is not enough. We have implemented the procedure CheckLoginAttempt 20. The value of MaxAttempAllowed should not be higher than 5. Each time where an unsuccessful attempt happen from the same session, a session counter will be incremented by 1. After the maximum threshold of unsuccessful attempts, the account should be locked out. The Brute Force Attack is very common, so this simple solution will help greatly to remove this vulnerability from a web application.

Algorithm 20 CheckLoginAttempt(MaxAttempAllowed)

```
1: procedure CHECKLOGINATTEMPT (MAXATTEMPALLOWED)
2:     DECLARE a SESSION Variable LoginCount
3:     Initialize Session("LoginCount") to 0 only at the first time of the session
4:     EACH TIME Login attempted increment the variable by 1
5:     Session("LoginCount") = Session("LoginCount") + 1
6:     if Session("LoginCount")> MaxAttempAllowed  then
7:         return ERROR with possible breach
8:     end if
9: end procedure
```

5.5.4 Insecure password policy

'The True Cost of Unusable Password Policies' [77] has described in great detail why Insecure Password Policy is unusable and vulnerable. Enforcing minimum length, stregth, combination of uppercase, lowercase and numbers ensure a strong password. There are algorithms and implementations in web spectrum under free license those can be directly plugged in web application. Example websites are http://metabetageek.com/2010/05/19/password-meter-version-2-0-now-available/ and http://www.passwordmeter.com/.

5.5.5 HTTPS not enforced

There are web applications where https is not mandatory throughout the site. The site doest not enforce https in the content area but force https in the transactional area. A good example is online ticketing solutions. For example you can browse http://barbican.org.uk up until someone click on book now button. Once you click book now button, you are automatically redirected to https. So enforcing needs to

be done page level on that scenario so that user cannot bypass https where you want them to be secure.

The routine HTTPRedirection 21 will automatically redirect user to https although the user types http in the browser. This logic also needs to check whether secure certificate is installed in the server. It is easy to check by server side syntax in web application. This simple solution will save the application from hackers where there are mixed type pages in the website.

Algorithm 21 HTTPRedirection

```
1: procedure HTTPREDIRECTION
2:     if REQUESTTED_SERVER_PORT <> 443 AND HTTPS_INSTALLED_IN_THE
   SERVER then
3:         REDIRECT THE REQUEST IN THE SAME PAGE by REPLACING "http"
   with "https"
4:     end if
5: end procedure
```

Chapter 6

Circular pattern matching for One to Many Fingerprint Identification

6.1 Introduction

The fingerprint identification is not a new topic but there is still a space for the enhancement[78]. Interesting enough, the proposed algorithm in [45] was the first attempt to utilise circular string matching techniques for solving fingerprints recognitions problem accurately and efficiently. The proposed algorithm was to convert the fingerprints image into circular strings mainly to solve the rotation problem. Later, in [79], the authors showed the high accuracy and efficiency of the algorithm by implementing it and analysing the experiment results.

In this chapter I will present a new technique to reduce search space of fingerprints by applying liner time filters on the saved fingerprints regardless of the different rotations of these fingerprints in the database. Based on the user's choice of implementations, this process may work as preprocessing step or can be embedded in real time. I will recommend this as a preprocessing stage so that in run-time there is not much efforts on calculating the filters.

The fingerprints in the database will be saved in a circular string format that will be extracted using the extraction algorithm in [45]. Afterwords, four effective filters

will be applied on the database and the result will be saved in the database row of respective fingerprint. Before running any costly matching algorithm, the best candidates from the database will be chosen by using the saved values of the filters.

6.2 Related works

When it comes to solving fingerprints identification problem with large databases, fingerprint indexing and fingerprint classification are the most reliable solutions until now [36]. Henry classifications system [37] classified the fingerprints into five categories: left loop, right loop, whorl, arch, and tented arch. Nevertheless, these number of categories are relativity small comparing to the number of the fingerprints in the databases. To explain, fingerprints indexing is to present the fingerprints with feature vectors. This approach is deploy on matching stage where a query fingerprints will be matched with the fingerprints in database which their vectors are similar to query vector [36]. Various algorithms have been proposed to improve fingerprints indexing which involve either global feature or minutiae feature. In [38] and [39] the proposed algorithms based on features of triangle or quadrangle geometric and selected simple hashing methods. However, these geometric characteristic have more sensitivity to distortion and noise. whereas, [40] designed a descriptor used a minutiae triplet, their features of triplet consists of the angles, the length of each length and the ridge to be calculated between each minutiae pair. However, a false rate might be happening due to that the minutia is superior in discrimination. A pairwise enrolment is used to reduce the false rate between each input fingerprint and the stored fingerprints in the database using clustering transformation parameter, however, this costs a lot of time. Later, [38] developed the algorithm in [40] to increase the matching accuracy and the speed. They used a new minutiae triplets feature and effective ad-hoc geometric rules on the theory of triplets matching instead of pairwise enrollment. Therefore, an logarithm has been proposed in [41] to use a binary Minutia Cylinder Code (MCC) to

be a minutia descriptor. This approach is proposed to encode the direction and location of neighbour minutiae around per minutia into a bit vector with fixed length. Although, MCC has higher accuracy and speed but it requires a higher dimension. Alternatively, Locality Sensitive Hashing (LSH) is utilised to search for MCC hypothesis correspondences rather than original quantisation strategy.However,it is not efficient for applications need high accuracy.

6.3 Problem definition

For a given fingerprint, I am given circular strings which is extracted from [45] and this needs to be identified in a large database. The given strings are essentially circular strings with binary alphabet. Here I consider the problem of finding occurrences of a pattern string $\mathcal{P}$ of length m with circular structure in a text string $\mathcal{T}$ of length n with linear structure and circular structures are binary strings. For instance, to find out the binary sequence of a circular structure, it must locate all positions in $\mathcal{T}$ where at least one rotation of $\mathcal{P}$ occurs. I also consider the given solutions on [45] for one to one fingerprint identification. Identifying the given fingerprint in a large database is the problem I have chosen to address.

6.4 Contribution

In this chapter I extended my work on circular pattern matching to identify fingerprint in a large database. My focus here is to reduce the search space by using effective filters those I explained in detail in Chapter 2 and Chapter3.

6.5 Preliminaries

I explained Circular pattern matching in great detail in previous chapters. I am just providing the basics here in fingerprint perspective where I need to use binary alphabet instead of genomic alphabet because the given circular strings consist of binary alphabet $\{0, 1\}$.

A circular string of length m can be viewed as a traditional linear string which has the left-most and right-most symbols wrapped around and stuck together in some way. Under this notion, the same circular string can be seen as m different linear strings, which would all be considered equivalent. Given a string $\mathcal{P}$ of length m, I denote by $\mathcal{P}^i = \mathcal{P}[i : m]\mathcal{P}[1 : i - 1]$, $0 < i < m$, the i-th *rotation* of $\mathcal{P}$ and $\mathcal{P}^0 = \mathcal{P}$.

I consider the binary alphabet, i.e., $\Sigma = \{0, 1\}$. In my approach, each character of the alphabet is presented by its numeric value. I use $num(x), x \in \Sigma$ to denote the numeric value of the character x. So, I have $num(0) = 0, num(1) = 1$. For a string S, I use the notation S_N to denote the numeric representation of the string S; and $S_N[i]$ denotes the numeric value of the character $S[i]$. So, if $S[i] = 1$ then $S_N[i] = num(1) = 1$. The concept of circular string and their rotations also apply naturally on their numeric representations as is illustrated in Example below.

6.5.1 Example

Suppose I have a pattern $\mathcal{P} = 0110101$. The numeric representation of $\mathcal{P}$ is $\mathcal{P}_N = 0110101$. And this numeric representation has the following rotations: $\mathcal{P}_N^1 = 1101010$, $\mathcal{P}_N^2 = 1010101$, $\mathcal{P}_N^3 = 0101011$, $\mathcal{P}_N^4 = 1010110$, $\mathcal{P}_N^5 = 0101101$, $\mathcal{P}_N^6 = 1011010$, $\mathcal{P}_N^7 = 0110101$.

The problem I handle in this chapter can be formally defined as follows.

6.5.2 Problem

(Approximate Circular Pattern Matching with k-mismatches (ACPM)). Given a pattern $\mathcal{P}$ of length m, a text $\mathcal{T}$ of length $n > m$, and an integer threshold $k < m$, find all factors $\mathcal{F}$ of $\mathcal{T}$ such that $\mathcal{F} \equiv_k \mathcal{P}^i$ for some $0 \leq i < m$. And when I have a factor $\mathcal{F} = \mathcal{T}[j : j + |\mathcal{F}| - 1]$ such that $\mathcal{F} \equiv_k \mathcal{P}^i$ I say that the circular pattern $\mathcal{C}(\mathcal{P})$ k-matches $\mathcal{T}$ at position j. I also say that this k-match is due to $\mathcal{P}^i$, i.e., the ith rotation of $\mathcal{P}$.

In this chapter I have solved one to many fingerprint problem by adapting my work [80] on circular pattern matching. Chapter 2 and Chapter 3 provide the detail explanation.

6.6 Filtering Algorithm

As it has been mentioned above, the algorithm is based on some filtering techniques. Suppose I am given a pattern $\mathcal{P}$ and a text $\mathcal{T}$. I will frequently and conveniently use the expression "$\mathcal{C}(\mathcal{P})$ k-matches $\mathcal{T}$ at position i" (or equivalently, "$\mathcal{P}$ circularly k-matches $\mathcal{T}$ at position i") to indicate that one of the conjugates of $\mathcal{P}$ k-matches $\mathcal{T}$ at position i (or equivalently, $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$).

6.6.1 Database filters

I employ a total of four filters. The key to the observations and the resulting filters is the fact that each function I devise results in an output when applied to the rotations of a circular string. I adapt 4 filters below based on binary alphabet $\{0, 1\}$.

6.6.2 Filter 1

I define the function sum on a string $\mathcal{P}$ of length m as follows: $sum(\mathcal{P}) = \sum_{i=1}^{m} P_N[i]$. The first filter, Filter 1, is based on this sum function. I have the following observation.

Observation 1 *Consider a circular string $\mathcal{P}$ and a linear string $\mathcal{T}$ both having length n. If $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$, where $0 \leq k < n$, then we must have*

$$sum(\mathcal{T}) - k \times 1 + k \times 0 \leq sum(\mathcal{C}(\mathcal{P})) \leq sum(\mathcal{T}) + k \times 1 - k \times 0.$$

6.6.3 Filter 2

Observation 2 *Consider a circular string $\mathcal{P}$ and a linear string $\mathcal{T}$ both having length n and assume that $\mathcal{A} = ext(\mathcal{P})$ and $\mathcal{B} = ext(\mathcal{T})$. If $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$, where $0 \leq k < n$, then I must have*

$$abs\_total\_distance(\mathcal{B}) - k \times 1 + k \times 0 \leq abs\_total\_distance(\mathcal{C}(\mathcal{A}))$$

$$\leq abs\_total\_distance(\mathcal{B}) + k \times 1 - k \times 0.$$

6.6.4 Filter 3

Now I present the following related observation which is the basis of the Filter 3. Note that Observation 2 differs with Observation 3 only through using the absolute version of the function used in the latter.

Observation 3 *Consider a circular string $\mathcal{P}$ and a linear string $\mathcal{T}$ both having length n and assume that $\mathcal{A} = ext(\mathcal{P})$ and $\mathcal{B} = ext(\mathcal{T})$. If $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$, where $0 \leq k < n$, then I must have*

$$total\_distance(\mathcal{B}) - k \times 1 + k \times 0 \leq total\_distance(\mathcal{C}(\mathcal{A}))$$

$$\leq total\_distance(\mathcal{B}) + k \times 1 - k \times 0.$$

6.6.5 Filter 4

Filter 4 uses the $sum()$ function used by Filter 1, albeit, in a slightly different way. In particular, it applies the $sum()$ function on individual characters. So, for $x \in \Sigma$ I define $sum_x(\mathcal{P}) = \sum_{1 \leq i \leq |\mathcal{P}|, \mathcal{P}[i]=x} P_N[i]$. Now I have the following observation.

Observation 4 *Consider a circular string $\mathcal{P}$ and a linear string $\mathcal{T}$ both having length n. If $\mathcal{C}(\mathcal{P}) \equiv_k \mathcal{T}$, where $0 \leq k < n$, then I must have*

$$sum_x(\mathcal{T}) - k \times num(x) \leq sum_x(\mathcal{C}(\mathcal{P})) \leq sum_x(\mathcal{T}) + k \times num(x)$$

for all $x \in \Sigma$.

For clarity, the 4 filters are illustrated in the below with the example in terms of binary alphabet. For the detail illustrations of the filters including example and explanations, please refer to Chapter 2 and Chapter 3 where explanations are given on genomic letters perspective.

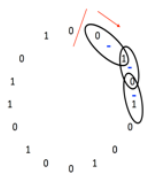
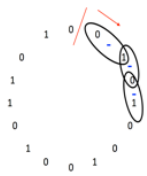
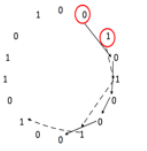
Circular Binary Image	Binary String p
	$Sum(p)$= 0+1+0+1+0+0+1+0+0+1+0+1+1+0+1+0=7
	$Sum(abs(distance(p)) =$ |(0-1)|+|(1-0)|+|(0-1)|+|(1-0)|+|(0-0)|+|(0-1)|+|(1-0)|+|(0-0)|+|(0-1)|+|(1-0)|+|(0-1)|+|(1-1)|+|(1-0)|+|(0-1)|+|(1-0)|+|0-0|=12
	$Sum((distance(p)) =$ (0-1)+(1-0)+(0-1)+(1-0)+(0-0)+(0-1)+(1-0)+(0-0)+(0-1)+(1-0)+(0-1)+(1-1)+(1-0)+(0-1)+(1-0)+(0-0)=0
	$Sum(idistance(0)) =$ 2+2+1+2+1+2+3+2+1=16 $Sum(idistance(1)) =$ 2+3+3+2+1+2+3=16

FIGURE 6.1: Filters by binary alphabet

Chapter 7

URL String Search in Steganography - Algorithm and Web Tools

7.1 Introduction

Steganography is the science of hiding data within data. I have provided background information on this topic in Chapter 1. In this Chapter, I propose a novel detection approach that concentrates on detecting hidden URLs in the loss-less images and extracting them from images. My approach uses the manipulations of Least Significant Bits [58] of the images which are imperceptible to human eyes.

7.2 Problem definition

The problem is to detect an URL hidden inside an image. As described in Chapter 1, any malicious code can be embedded by using least significant bit(s). Modifying LSB means modifying the colour by using least significant bits of an image. There are different type of colour formats such as 8 bits, 24 bits etc. They have both colour and grey scale. 8 bits colour means each pixel can have any of 256 (2^8) colour. The same calculation is applicable 8 bits grey scale or 24 bit colours. So modifying the

least significant bit in an array of huge combination of colours does not make much difference in human eye. This makes the use of LSB URL attack.

For example, an url http://exampleattack.com has 24 characters. Each character of this url takes 8 bits in ASCII format. The URL thus requires 192 bits from an image to encode.

For the simplicity of example, let us see how first character 'h' of our example url http://exampleattack.com can be added by using least significant bits of an image. The ASCII value for 'h' is decimal 104 and binary 01101000.

Before LSB insertion let us assmume that 8 consecutive bytes of an images is below.

10000010 10100110 11110101 10110101 10110011 10010111 10000100 10110001

After inserting 'h' (01101000) in least significant bits the result is below.

1000000**0** 1010011**1** 1111010**1** 1011010**0** 1011001**1** 1001011**0** 1000010**0** 1011000**0**

In this way by using more significant bit of images we can embed the rest of the characters of the intended URL.

7.3 Contribution

I have dealt with hidden URL detection in the image and explained the approach as well as provided with the algorithm and Pseudocode. The algorithm has also been implemented. Furthermore, the URL detection problem in an image has been simplified with respect to string matching approach which can be adapted in other kind of string matching problems in an image. For example, users may be interested to search for malicious commands or other kind of strings hidden in the image using least significant bits (LSB) of the image. In this chapter, I have implemented this algorithm and successfully tested and compared various results using different images.

7.4 URL detection algorithm

I am going to present an Algorithm overview in 7.4.1 and detail Psuedocode in 7.4.2 to detect a hidden URL from the least significant bits of an image. The detail complexity analyses is done in 7.4.3

AAA	AARP	ABB	ABBOTT	ABOGADO
AC	ACADEMY	ACCENTURE	ACCOUNTANT	ACCOUNTANTS
ACO	ACTIVE	ACTOR	AD	ADS
ADULT	AE	AEG	AERO	AF
AFL	AG	AGENCY	AI	AIG
AIRFORCE	AIRTEL	AL	ALLFINANZ	ALSACE
AM	AMICA	AMSTERDAM	ANALYTICS	ANDROID
AO	APARTMENTS	APP	APPLE	AQ
AQUARELLE	AR	ARAMCO	ARCHI	ARMY
ARPA	ARTE	AS	ASIA	ASSOCIATES
AT	ATTORNEY	AU	AUCTION	AUDI
AUDIO	AUTHOR	AUTO	AUTOS	AW
AX	AXA	AZ	AZURE	..etc

TABLE 7.1: List of Top-Level Domains by the ICANN - for full list please refer to [5]

7.4.1 Algorithm overview

Step 1: Create a sorted list, DOMAIN[], from the static official top level domain list.

Step 2: Create an array called BITMAP[], from an image taking each bit in array.

Step 3: Make a character array called, LSBCHARARRAY[] from an Intermediate array of LSBARRAY[] by converting each 8 bits to an ASCII character.

Step 4: Loop throuth the LSBCHARARRAY[], find out possible hidden url forming by http or https, www, domain name and top level domain(TLD).

7.4.2 The algorithm in Pseodocode

The 4 steps given in 5.1 are presented here with detail pseudocode so that users can convert any any programming language easily with little efforts.

```
1: procedure FindURLInImage
2:     CREATE a sorted indexed array DOMAIN[] from the official top level domain
   list
3:     CREATE a BITMAP[] array from the image taking each bit
4:     *Comment: Loop through the BITMAP[] and create an array LSBARRAY[]
   with the least significant bits
5:     Integer i, j
6:     i=0
7:     j=0
8:     for  i = 0 to BITMAP[] do
9:         if (i != 0) AND (i+1) MOD 8 = 0 then
10:            return LSBARRAY[j++] = BITMAP[i]
11:        end if
12:    end for
13:    *Comment: Loop through LSBARRAY[] and convert to a LSBCHARARRAY[]
   character array
14:    i=0
15:    j=0
16:    String t=""
17:    for  i = 0 to LSBARRAY[] do t = STRING((t) + LSBARRAY[i])
18:        if (i!=0) and (i+1) MOD 8 = 0 then
19:            return LSBCHARARRAY[j++] = ConvertToCharacter(t)
20:            t = ""
21:        end if
22:    end for
23:    *Comment: Loop through the LSBCHARARRAY[] to detect URL by using the
   DOMAIN[] array Integer temp, l,s
24:    ' Initialize i and s outrside the loop
25:    i=0
26:    s=0
27:    Boolean httpOrHttpsExists
28:    Boolean wwwExists
```

```
35:         wwwExists = False
36:         urlFound = False
37:         URL = ""
38:         J=0
39:         t=""
40:         temp = 0
41:         if LSBCHARARRAY[i] = ":" then
42:             *Comment:Check Possibility of having an http:\\
43:             t = ConvertToString(LSBCHARARRAY[i-4] to LSBCHARARRAY[i+2])
44:             if LowerCase(t) = "http:\\" then
45:                 httpOrHttpsExists = True
46:                  temp = i + 3
47:                 URL= "http:\\"
48:             end if
49:             t = ""
50:             if httpOrHttpsExists = False then
51:                 *Comment:Possibility of having an https:\\
52:                   t = ConvertToString(LSBCHARARRAY[i-5] to LSBCHARAR-
    RAY[i+2])
53:                 if LowerCase(t) = "https:
54: " then
55:                     httpOrHttpsExists = True
56:                     temp = i + 3
57:                     URL= "https:\\"
58:                 end if
59:             end if
60:              t = ""
61:             t = ConvertToString(LSBCHARARRAY[i+3] to LSBCHARARRAY[i+6])
62:             if LowerCase(t) = "www." then
63:                 temp = temp+ i + 5
64:                 wwwExists = True
65:                 URL= Concat(URL,"www.")
```

Algorithm 22 Procedure FindURLInImage

```
86:                 URL = Concat(URL, ConvertToString(LSBCHARARRAY[j-i+1]
    to LSBCHARARRAY[j]))
87:                 i = j + 1
88:                 urlFound = True
89:                 Exit FOR
90:             end if
91:         end for
92:         if urlFound then
93:             urlFound = False
94:             for j = i to LSBCHARARRAY[] do
95:                 t = Concat(t,LSBCHARARRAY[j])
96:                 *Comment: Now check t in sorted top level domain list DO-
    MAIN
97:                 if t EXISTS in DOMAIN[] then
98:                     URL = Concat(URL, ConvertToString(LSBCHARARRAY[j-
    i+1] to LSBCHARARRAY[j]))
99:                     urlFound = True
100:                    *Comment: Reinitialize the value of i for the next iteration
101:                    i = j + 1
102:                    EXIT FOR
103:                end if
104:            end for
105:        end if
106:    end if
107:    if urlFound then
108:        *Comment: It is possible to have multiple URL in different position
109:        OutPutURLArray[s] = URL
110:        s = s + 1
111:    end if
112: end for
113: if s > 0 THEN then
114:    *Comment: URL has been found and OutPutURLArray[] contains the urls
```

7.4.3 Complexity analyses

Step 1: Create a sorted list from the static official top level domain

Space complexity: I have a known TLD list [5]. So in the preprocessing stage, I create an indexed array, DOMIAN[] considering each TLD as a string. Space complexity is linear to the size of all characters plus the index of each string position in a sorted order. Also I create a separate index list with just starting position of TLDs with a specific character. For example, .co and .com both start with c, so if we know where the c starts in the whole sorted list, we just can look in the block starting with 'c'. The overall space complexity for the sorted list is O(M) +O(t) + O(i) where M is the total number of characters, t is the index on each TLD string which is limited to the official static list.

Time complexity: This step of computation is part of preprocessing, so complexity is not a major issue. However it is possible to build up sorted list by radix sort [81] where an LSD radix sort operates in O(nk) in all cases, where n is the number of keys, and k is the average key length.

Step 2: Create a sorted list from the static official top level domain list

Space complexity: O(M) where M is the number of bits.

Time complexity: O(n) where n is the number of bits. This means in just single iteration the array is built.

Step 3 : Make a character array by converting each 8 bits to an ASCII character

Space complexity: The complexity is O(n) here where n=M/8 where M is the number of bits in BitMap and only one in each 8 bits are placed in character array by converting 8 such Least Significant bits into character. So the complexity here is sub linear. Although an intermediate LSBARRAY has been introduced in Step 3 for clarity purpose of the flow, it is possible to calculate the LSBCHARARRAY directly from BITMAP[] array. So LSBARRAY[] is not required in the implementation.

Time complexity. This is looping through the BitMap array just once and producing character array by taking each 8 significant bits together and converting to ASCII. So the time complexity is linear here with O(n) where looping n bits just once produces the result. Converting to ASCII and character happens just 1 in 64 where 1 byte(8 consecutive LSB) comes from 64 bits. This operation produces time complexity of O(n+n/64) which is linear.

Step 4: Loop throuth the array, find out possible hidden url forming by http or https, www, domain name and top level domain(TLD)

Space complexity: The space complexity holds the linerity here with O(n) where n is the number of characters in the array.

Time complexity: This is a loop through the characters array. Finding first 3 parts of an URL (http/https and/or www, domaion name) are done in one go in the single loop. There are inside loops used to find the position and calculation purpose for http, https and www. The actual counter of characters array is incremented in each go whether it is inner loop or outer loop. The complexity holds linear for the operations because the whole characters array are traversed just once. Looking up the 4th part, Top Level Domain (TLD) requires a short look up in a sorted array described in Step 1. For the whole character array, this look up is just done to complete the search in a sorted and indexed Top Level Domain array which I called in step 1 as DOMAIN[]. In a sorted list, the binary search works as log(n) complexity in worst case where n is the number of items in an array. But in our case, n is narrowed down by index of each character. So the each block of searched area is n/m where m is the number characters in alphabet. So the search takes log(n/m)time because we know the starting character what to look up DOMAIN[] array. The overall complexity stays linear for step 4.

7.5 Experiments

I implemented this algorithm using Visual Studio 2015 Studio, ASP.Net 4.5 and Javascript. The solution is available on http://tanvera-001-site1.htempurl.com and the source code can be downloaded from http://www.samirsoftware.com/stegano.zip. The program source code link is available at 9 too. I have tested the solution using bmp and png images of different sizes, colour depth, colour palettes and compression types. The solution has been tested using IE11, Firefox 4 and Chrome Ver 50.0 . The solution needs to access files from client machines or folders, so if there are restrictions on accessing images files, the browser will not be able to read the images files. The URL in the experiments is restricted to the top level domain name. As it is the objective to keep the output image identical to the original image from the viewers prospective, the solution can only accept certain types of images as input. It cannot accept compressed and lossy images as there is a possibility that the URL data will be lost when the images are compressed and we will not be able to extract the URL from the stego image [57]. For monochrome images changing the LSB might alter the image in such a way that the changes are visible to the viewers and raise suspicion that the image have been altered.

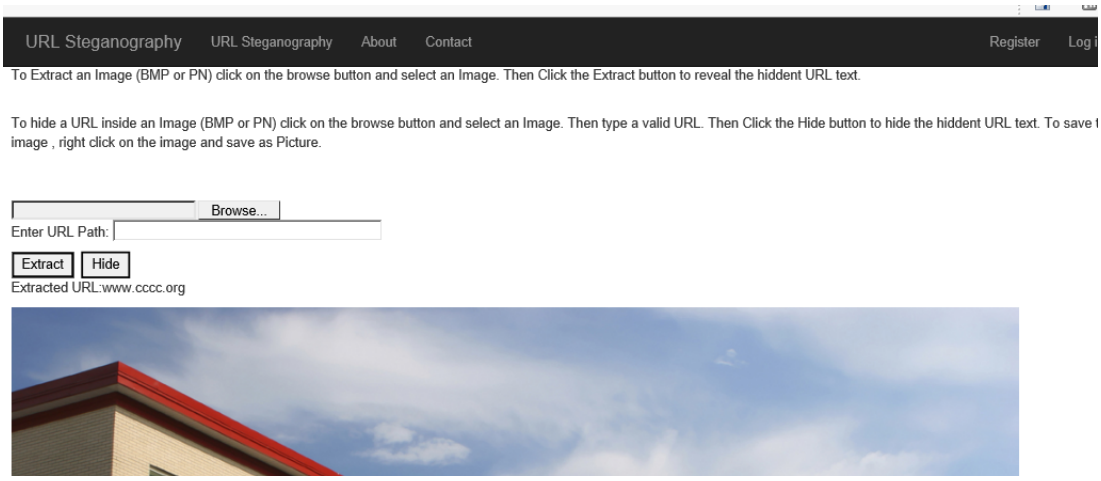


FIGURE 7.1: Extracting URL from image

Extracting URL Images: The users needs to select the image using the "Browse" button , which will open a file selector window. The user needs to select a bmp, gif or png image. Once the image file is selected then the user needs to click "Extract URL" button. The website will then upload the image in the server, extract the bitmap of the image and apply the algorithm. If it identifies the URL then it will display the output on the screen 7.1. If an URL is not present, then an appropriate message is displayed.

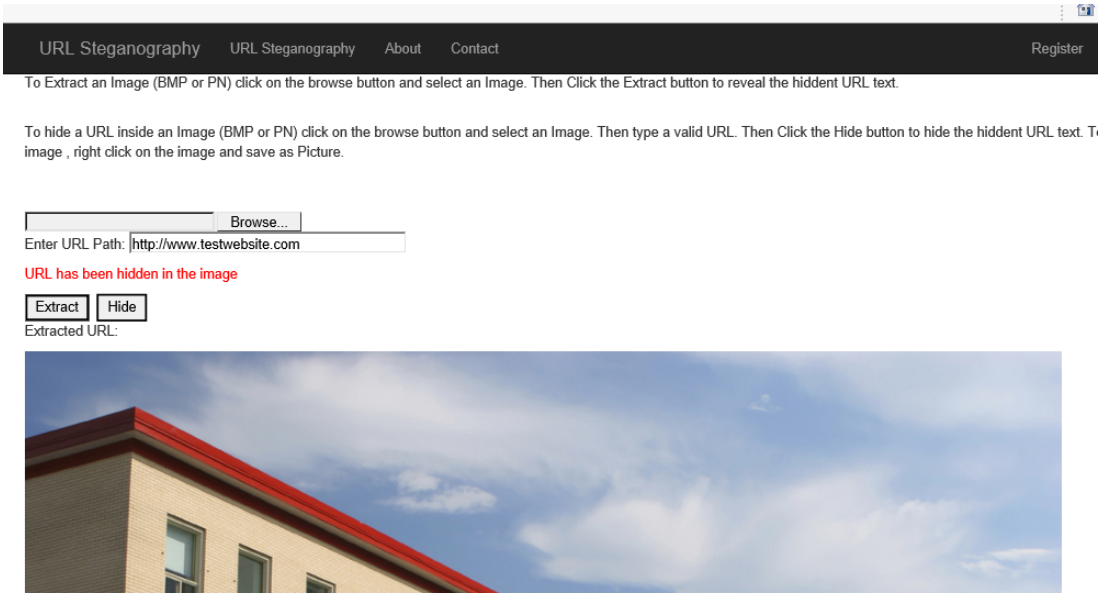


FIGURE 7.2: Hidding URL using stego process

Hiding URL Images: The user need to select the image where they want to hide the URL using the "Browse" button, which will open a file selector window. The user needs to select a bmp, gif or png image. Once the image file is selected then the URL needs to be entered in the given text box. The user needs to click "Hide URL" button. The system will first verify the URL is a valid URL , otherwise it will display an error message. The website will then upload the image in the server, extract the bitmap of the image and hide the URL inside the image. The modified image will be then displayed on the browser 7.2. The user can right click on the image and save the image in a desired location.

7.5.1 Checking experiment results

7.5.2 Image difference

I tested the generated images with the original using a free Image comparison website [82]. The website found no difference between the original and image containing the hidden URL 7.3. It compares the pixel value and colour between the images , there is a threshold (3 points) which the pixel must exceed in order to register as a difference.

FIGURE 7.3: Image difference

7.5.3 Histograms analysis

I have analyzed the histograms of the image and the generated stego image. There was no difference between the histograms of both these original image 7.4 and the stego image 7.5.

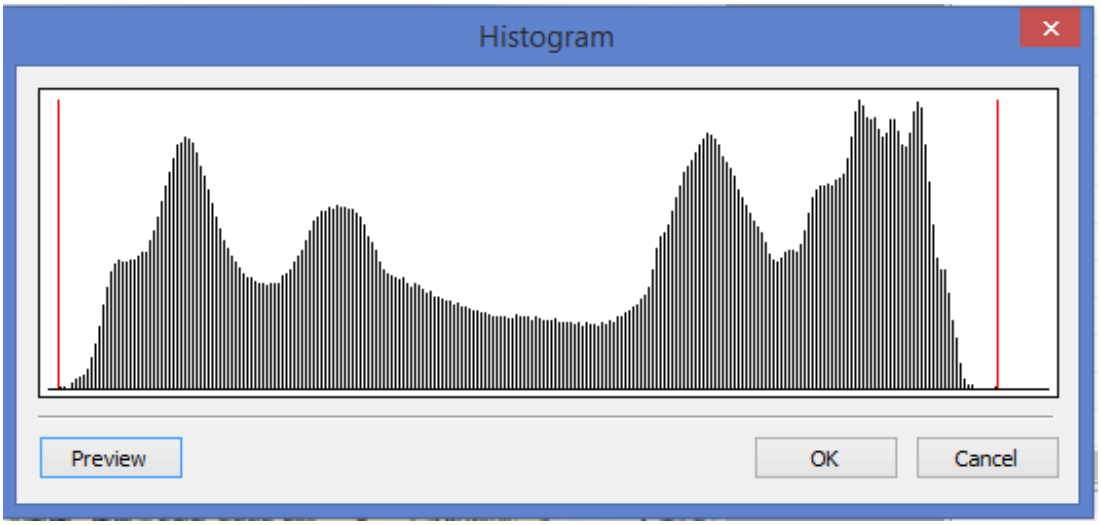


FIGURE 7.4: Histograms Analysis

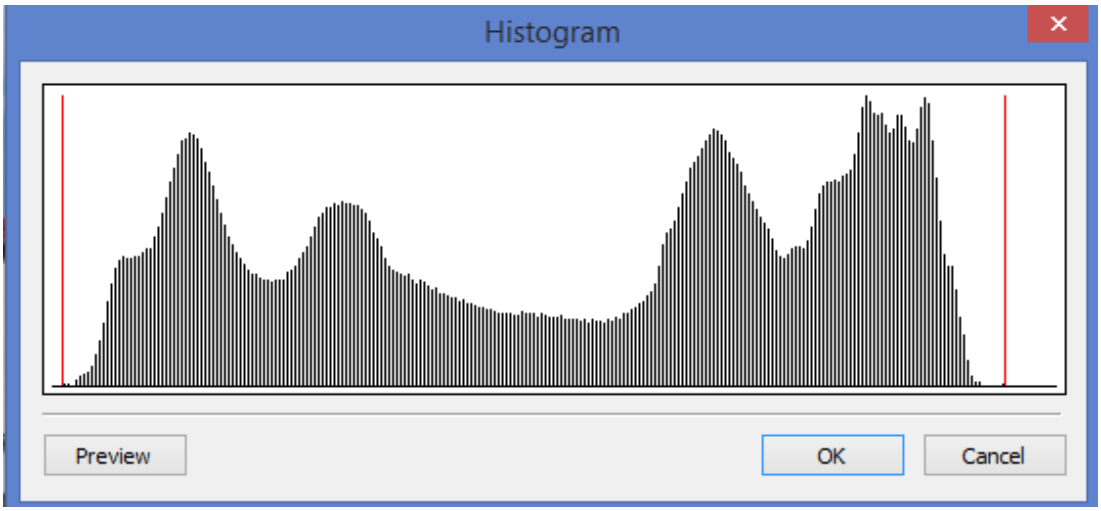


FIGURE 7.5: Histograms analysis in stego image

Chapter 8

Conclusion

8.1 Contribution and future direction

I have developed efficient algorithms and implemented web based tool on circular pattern matching. I have applied circular pattern matching algorithms to solve One to Many fingerprint problems. Web based tools attracts the inherent web vulnerabilities, those I have addressed by providing the solutions in both algorithmic and practical perspective. I have done further work by providing algorithm and implementations on web security in Steganography perspective.

In Chapter 2 and Chapter 3, I have employed effective lightweight filtering technique to reduce the search space of the Circular Pattern Matching (CPM) problem. I have conducted experiments to show the effectiveness of the resulting algorithm based on the filters for both exact and approximate Circular Pattern Matching problem. I compared the results with the state of the art algorithms and showed the effectiveness of my filters and algorithms. I have presented that the elapsed time of my algorithms are much smaller than the state of the art algorithm[4] for the filtered text string. I have presented SimpLiFiACPM, an extremely fast algorithm based on the filters for approximate circular pattern matching. Much of the speed of this algorithm comes from the fact that the filters are effective but extremely simple and lightweight. The most intriguing feature of SimpLiFiACPM is perhaps its capability to plug in any algorithm to solve ACPM and take advantage of it. My algorithm turns out to be

much faster in practice because of the huge reduction in the search space through filtering. In Chapter 6, I have provided a uses cases for Circular Pattern Matching to solve One to Many fingerprints identification problem. The presented approach emphasised on reducing the search space of the circular pattern matching problem by deploying effective lightweight filtering technique in a linear time.

The future researcher can find out more application areas of Circular Pattern Matching and solve those in practical perspective by using the algorithms proposed in this thesis.

In Chapter 4 I developed a fast and efficient browser based tool based on the algorithms provided in Chapter 2 and Chapter 3. To my knowledge while this thesis is underway, this is the first web based Circular Pattern Matching tool that operates in client side. I have conducted a good amount of experiments in Chapter 2 and Chapter 3, that gives an excellent academic framework of this work which leads to the development of web based practical tool on Circular Pattern Matching. Also the selection of technology, design patterns and the approach of development comprised a professional standard so that this tool can set an example in terms of methodology of developing similar kind of software in academic arena. The users do not need to install any software or do not need to upload the big file in the server. My development approach works with big data in client side by using lower memory working as chunk by chunk without uploading any data to web server. Instead of program running in the web server, the java script code, is transported in the browser in runtime and does the computation. Inherently any upload over http is slow. Also in server side, various factors such as LAN speed, server's concurrent users, speed on various hops over Internet up to server, influence the upload of big file. For example, Blast (http://blast.ncbi.nlm.nih.gov/Blast.cgi) is a great tool for bioinformaticians but I have a scenario where I need to upload both query and reference files to the server, then it is very difficult for big file to upload over http to Blast. I have been able to upload the highest 600MB from both my home and college infrastructure to blast after

several tries. In most cases it resulted by losing connection session with server or by crashing browser while uploading big data. So I developed my tool by using client side programming language so that it does not fall in the upload trap over http. Many sites open a dedicated FTP support for the users so that big data can be uploaded. But that means the computation does not happen in real time and manual procedure is needed to complete the operations. It is understandable that for many other applications, a huge server side resource is necessary, but surely those application should provide different architecture such as socket or ftp, rather than http. My main aim is to produce result for big chunk of data in the browser with possible minimum waiting time for the user. I computed my CPM algorithm on 3 GB data at ease but it can easily work with large data because the memory does not load the data at a time. It works chunk by chunk.

There is an interesting area on this tools where the future works can be done. When a client program needs huge computational resource such as a big server's computational power, then a new approach will have to be taken to resolve the issue. A clever program needs to be written by AJAX so that only necessary data is transferred in runtime and computation is done in combination of client and server machine.

In Chapter 5, I presented guidelines and solutions for web vulnerabilities. The proposed solutions are described in separate sections in an algorithmic manner. The solutions came across by learning the reasons of vulnerability[47] and penetration technique used by the scanning tools [48] to exploit the vulnerability. The vulnerability details have been described briefly to cover the basics of common vulnerabilities. I have given detail description of the solutions. A very brief overview of each problem and the solutions with the best practices as well as PSEUDO-CODE has been provided in this thesis. The relevant example codes with vbscript and Javascript has been uploaded in the url [65]. The Javascript code I have provided can be plugged directly in web application at the client side because Javascript is de facto standard of browser based client side language.

Web security is an area where constant improvement of the techniques is needed. The future researcher can look at the guidelines and solutions given in this thesis on web vulnerabilities. They can work on improving the solution in light of new vulnerabilities and threats.

In Chapter 5,I have dealt with hidden URL detection in the image and explained the approach as well as provided with the algorithm and Pseudocode. The algorithm has also been implemented. Furthermore, the URL detection problem in an image has been simplified with respect to string matching approach which can be adapted in other kind of string matching problem in an image. For example, users may be interested to search for malicious commands or other kind of strings hidden in the image using least significant bits (LSB) of the image. In this chapter, I have implemented this algorithm and successfully tested and compared various results using different images.

In a nutshell, this thesis provided efficient algorithms and implementation on few areas of applications. This thesis provided a professional standard web based tool along with the detail on web tool vulnerabilities as well as security. In the web tool, a state of the art client side technique has been used which is first in nature on Circular Pattern Matching problem while this thesis is underway. In general the ideas can be used by future researcher to devise efficient algorithms and techniques. The programming codes and technique used in this thesis can be used by future researchers to solve their use cases.

Chapter 9

Appendix

9.1 Circular pattern matching

The program source code for Circular pattern matching is available at
http://www.samirsoftware.com/CPM.zip

9.2 Approximate circular pattern matching

The program source code for Approximate circular pattern matching is available at http://www.samirsoftware.com/ACPM.zip

9.3 Web tool for approximate circular pattern matching

The program source code of Web tool for Approximate Circular Pattern Matching is available at http://www.samirsoftware.com/tool.zip

9.4 Web security

The web security guidance source code is available at
http://www.samirsoftware.com/code.pdf

9.5 Steganography

The program source code for Steganography tool is available at
http://www.samirsoftware.com/stegano.zip

9.6 ACPM result analysis

Timing for 720 permutations of 6 filters are available at
http://www.samirsoftware.com/resultAnalysis.xlsx

Bibliography

[1] Shashank Gupta and BB Gupta. "Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art". In: *International Journal of System Assurance Engineering and Management* (2015), pp. 1–19.

[2] Hossain Shahriar and Mohammad Zulkernine. "Client-side detection of cross-site request forgery attacks". In: *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*. IEEE. 2010, pp. 358–367.

[3] Md. Aashikur Rahman Azim et al. "A fast and lightweight filter-based algorithm for circular pattern matching". In: *BIBM HPCB* (2014).

[4] Carl Barton, Costas Iliopoulos, and Solon Pissis. "Fast algorithms for approximate circular string matching". In: *Algorithms for Molecular Biology* 9.1 (2014), p. 9. ISSN: 1748-7188. DOI: 10.1186/1748-7188-9-9. URL: http://www.almob.org/content/9/1/9.

[5] ICANN. *List of Top-Level Domains*. 2016. URL: https://www.icann.org/resources/pages/tlds-2012-02-25-en.

[6] M. Lothaire. In: *Applied Combinatorics on Words, New York, NY, USA: Cambridge University Press* (2005).

[7] K Fredriksson and S Grabowski. "Average-optimal string matching". In: *J Discrete Algorithms* 7.4 (2009), pp. 579–594. DOI: 10.1016/j.jda.2008.09.001.

[8] KH Chen, GS Huang, and RCT Lee. "Bit-parallel algorithms for exact circular string matching". In: *Comput J* (2013). doi:10.1093/comjnl/bxt023.

[9] Md. Aashikur Rahman Azim et al. "A fast and lightweight filter-based algorithm for circular pattern matching". In: *ACM BCB* (2014).

[10] Md. Aashikur Rahman Azim et al. "SimpLiFiCPM: A Simple and Lightweight Filter-Based Algorithm for Circular Pattern Matching". In: *International Journal of Genomics* (), Volume 2015 (2015), Article ID 259320.

[11] Md. Aashikur Rahman Azim et al. "A Filter-Based Approach for Approximate Circular Pattern Matching". In: *Bioinformatics Research and Applications, ISBRA 2015, Norfolk, VA, USA, June 7-10, 2015 Proceedings* (2015), pp. 24–35.

[12] M. A. R. Azim et al. "A Simple, Fast, Filter-Based Algorithm for Approximate Circular Pattern Matching". In: *IEEE Transactions on NanoBioscience* 15.2 (2016), pp. 93–100. ISSN: 1536-1241. DOI: 10.1109/TNB.2016.2542062.

[13] http://www.inf.kcl.ac.uk/research/projects/asmf/.

[14] CS Iliopoulos and MS Rahman. "Indexing circular patterns". In: *Proceedings of the 2nd International Conference on Algorithms and Computation* (2008), pp. 46–57.

[15] J Lin and D Adjeroh. "All-against-all circular pattern matching". In: *Comput J* 55.7 (2012), pp. 897–906. DOI: 10.1093/comjnl/bxr126.

[16] R Weil and J Vinograd. "The cyclic helix and cyclic coil forms of polyoma viral DNA". In: *Proc Natl Acad Sci* 50.4 (1963), pp. 730–738. DOI: 10.1073/pnas.50.4.730.

[17] R Dulbecco and M Vogt. "Evidence for a ring structure of polyoma virus DNA". In: *Proc Natl Acad Sci* 50.2 (1963), pp. 236–243. DOI: 10.1073/pnas.50.2.236.

[18] M Thanbichler, SC Wang, and L Shapiro. "The bacterial nucleoid: A highly organized and dynamic structure". In: *J Cell Biochem* 96.3 (2005). [http://dx.doi.org/10.1002/jcb.205 pp. 506–521. DOI: 10.1002/jcb.20519.

[19] G Lipps. In: *Plasmids: Current Research and Future Trends* (2008).

[20] T Allers and M Mevarech. "Archaeal genetics – the third way". In: *Nat Rev Genet* 6 (2005), pp. 58–73. DOI: 10.1038/nrg1504.

[21] D Gusfield. In: *Algorithms on Strings, Trees and Sequences* (1997).

[22] A Mosig et al. "Comparative analysis of cyclic sequences: viroids and other small circular RNAs". In: *German Conference on Bioinformatics, Volume 83 of LNI* (2006), pp. 93–102.

[23] F Fernandes, L Pereira, and A Freitas. "CSA: An efficient algorithm to improve circular DNA multiple alignment". In: *BMC Bioinformatics* 10 (2009), pp. 1–13. DOI: 10.1186/1471-2105-10-1.

[24] T Lee et al. "Finding optimal alignment and consensus of circular strings". In: *Proceedings of the 21st annual Conference on Combinatorial Pattern Matching* (2010), pp. 310–322.

[25] Davide Maltoni et al. *Handbook of fingerprint recognition*. Springer Science & Business Media, 2009.

[26] Manhua Liu, Xudong Jiang, and Alex Chichung Kot. "Efficient fingerprint search based on database clustering". In: *Pattern Recognition* 40.6 (2007), pp. 1793–1803.

[27] Manhua Liu and Pew-Thian Yap. "Invariant representation of orientation fields for fingerprint indexing". In: *Pattern Recognition* 45.7 (2012), pp. 2532–2542.

[28] Raffaele Cappelli and Matteo Ferrara. "A fingerprint retrieval system based on level-1 and level-2 features". In: *Expert Systems with Applications* 39.12 (2012), pp. 10465–10478.

[29] Alessandra A Paulino et al. "Latent fingerprint indexing: Fusion of level 1 and level 2 features". In: *Biometrics: Theory, Applications and Systems (BTAS), 2013 IEEE Sixth International Conference on*. IEEE. 2013, pp. 1–8.

[30] Daniel Peralta et al. "Fast fingerprint identification for large databases". In: *Pattern Recognition* 47.2 (2014), pp. 588–602.

[31] Qinzhi Zhang and Hong Yan. "Fingerprint classification based on extraction and analysis of singularities and pseudo ridges". In: *Pattern Recognition* 37.11 (2004), pp. 2233–2243.

[32] E. R. Henry. *Classification and Uses of Finger Prints*. Routledge, 1900.

[33] Henry C. Lee, Robert Ramotowski, and R. E. Gaensslen, eds. *Advances in Fingerprint Technology, Second Edition*. CRC Press, 2002.

[34] Sruthy Sebastian. "Literature survey on automated person identification techniques". In: *International Journal of Computer Science and Mobile Computing* 2.5 (2013), pp. 232–237.

[35] Davide Maltoni et al. *Handbook of Fingerprint Recognition*. Springer-Verlag, 2009.

[36] Chaochao Bai et al. "An Efficient Indexing Scheme Based on K-Plet Representation for Fingerprint Database". In: *Intelligent Computing Theories and Methodologies*. Springer, 2015, pp. 247–257.

[37] Edward Richard Henry. *Classification and uses of finger prints*. HM Stationery Office, 1905.

[38] Bir Bhanu and Xuejun Tan. "Fingerprint indexing based on novel features of minutiae triplets". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 25.5 (2003), pp. 616–622.

[39] Ogechukwu Iloanusi, Aglika Gyaourova, and Arun Ross. "Indexing fingerprints using minutiae quadruplets". In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*. IEEE. 2011, pp. 127–133.

[40] Robert S Germain, Andrea Califano, and Scott Colville. "Fingerprint matching using transformation parameter clustering". In: *Computing in Science & Engineering* 4 (1997), pp. 42–49.

[41] Raffaele Cappelli, Matteo Ferrara, and Davide Maltoni. "Fingerprint indexing based on minutia cylinder-code". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 33.5 (2011), pp. 1051–1057.

[42] Tong Liu et al. "Fingerprint indexing based on singular point correlation". In: *Image Processing, 2005. ICIP 2005. IEEE International Conference on*. Vol. 3. IEEE. 2005, pp. II–293.

[43] Manhua Liu, Xudong Jiang, and Alex Chichung Kot. "Fingerprint retrieval by complex filter responses". In: *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*. Vol. 1. IEEE. 2006, pp. 1042–1042.

[44] Yi Wang, Jiankun Hu, and Damien Phillips. "A fingerprint orientation model based on 2D Fourier expansion (FOMFE) and its application to singular-point detection and fingerprint indexing". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 29.4 (2007), pp. 573–585.

[45] AL-Jamea Moudhi et al. "A Novel Pattern Matching Approach for Fingerprint-based Authentication". In: ().

[46] WhiteHat. *WhiteHat Security Statistics Report 2015*. Available at https://www.whitehatsec.com/statistics-report/featured/2015/05/21/statsreport.html. 2015.

[47] CWE/SANS. *CWE/SANS Top 25 Most Dangerous Software Errors*. Available at http://cwe.mitre.org/top25/archive/2011/2011_cwe_sans_top25.pdf. 2011.

[48] Nuno Antunes and Marco Vieira. "Benchmarking vulnerability detection tools for web services". In: *Web Services (ICWS), 2010 IEEE International Conference on*. IEEE. 2010, pp. 203–210.

[49] *Stegoloader: A Stealthy Information Stealer*. 2013. URL: https://www.owasp.org/index.php/Top10#tab=OWASP_Top_10_for_2013/.

[50] Mehdi Hariri, Ronak Karimi, and Masoud Nosrati. "An introduction to steganography methods". In: *World Applied Programming* 1.3 (2011), pp. 191–195.

[51] Robert Krenn. "Steganography and steganalysis". In: *Retrieved September* 8 (2004), p. 2007.

[52] Neil F Johnson and Sushil Jajodia. "Exploring steganography: Seeing the unseen". In: *Computer* 31.2 (1998), pp. 26–34.

[53] Niels Provos and Peter Honeyman. "Hide and seek: An introduction to steganography". In: *Security & Privacy, IEEE* 1.3 (2003), pp. 32–44.

[54] Abbas Cheddad et al. "Digital image steganography: Survey and analysis of current methods". In: *Signal processing* 90.3 (2010), pp. 727–752.

[55] softonic. *Xiao Steganography*. 2015. URL: http://xiao-steganography.en.softonic.com/.

[56] Chandan Mohapatra and Manjusha Pandey. "A Review on current Methods and application of Digital image Steganography." In: *International Journal of Multidisciplinary Approach & Studies* 2.2 (2015).

[57] Tayana Morkel, Jan HP Eloff, and Martin S Olivier. "An overview of image steganography." In: *ISSA*. 2005, pp. 1–11.

[58] Yambem Jina Chanu, Themrichon Tuithung, and Kh Manglem Singh. "A short survey on image steganography and steganalysis techniques". In: *Emerging Trends and Applications in Computer Science (NCETACS), 2012 3rd National Conference on*. IEEE. 2012, pp. 52–55.

[59] O. Kendirli E. Satir. "A Distortionless Image Steganography Method via URL". In: *The 7th International Conference Information Security and Cryptology*. 2014.

[60] Dell SecureWorks Counter Threat Unit™ Threat Intelligence. *Stegoloader: A Stealthy Information Stealer*. 2015. URL: http://www.secureworks.com/cyber-

threat-intelligence/threats/stegoloader-a-stealthy-information-stealer/.

[61] Dell SecureWorks Counter Threat Unit Brett Stone-Gross Ph.D. *Malware Analysis of the Lurk Downloader*. 2014. URL: http://www.secureworks.com/cyber-threat-intelligence/threats/malware-analysis-of-the-lurk-downloader/?view=Standard.

[62] http://hgdownload-test.cse.ucsc.edu/goldenPath/hg19/bigZips/.

[63] http://samirsoftware.com/acpm/index.html.

[64] https://www.samirsoftware.com/Tool.zip.

[65] M Samiruzzaman. *Vulnerability fix example in vbscript and javascript*. Available at http://samirsoftware.com/code.pdf.

[66] Microsoft TechNet. *SQL Injection*. Available at https://technet.microsoft.com/en-us/library/ms161953(v=SQL.105).aspx.

[67] Stephen W Boyd and Angelos D Keromytis. "SQLrand: Preventing SQL injection attacks". In: *Applied Cryptography and Network Security*. Springer. 2004, pp. 292–302.

[68] WG Halfond, Jeremy Viegas, and Alessandro Orso. "A classification of SQL-injection attacks and countermeasures". In: *Proceedings of the IEEE International Symposium on Secure Software Engineering*. Vol. 1. IEEE. 2006, pp. 13–15.

[69] Isatou Hydara et al. "Current state of research on cross-site scripting (XSS)–A systematic literature review". In: *Information and Software Technology* 58 (2015), pp. 170–186.

[70] Giuseppe A Di Lucca et al. "Identifying cross site scripting vulnerabilities in web applications". In: *Telecommunications Energy Conference, 2004. INTELEC 2004. 26th Annual International*. IEEE. 2004, pp. 71–80.

[71] Nayeem Khan, Johari Abdullah, and Adnan Shahid Khan. "Towards vulnerability prevention model for web browser using interceptor approach". In: *IT in Asia (CITA), 2015 9th International Conference on*. IEEE. 2015, pp. 1–5.

[72] Isatou Hydara et al. "Current state of research on cross-site scripting (XSS)–A systematic literature review". In: *Information and Software Technology* 58 (2015), pp. 170–186.

[73] Mohd Shadab Siddiqui and Deepanker Verma. "Cross site request forgery: A common web application weakness". In: *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*. IEEE. 2011, pp. 538–543.

[74] OWASP. *Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet*. Available at https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet. 2015.

[75] Konstantin Käfer. *Cross site request forgery*. 2008.

[76] *Setting up secure cookies*. Available at https://msdn.microsoft.com/en-us/library/ms228262(v=VS.80).aspx.

[77] Philip Inglesant M. Angela Sasse. *The True Cost of Unusable Password Policies:Password Use in the Wild*. Available at https://www.cl.cam.ac.uk/~rja14/shb10/angela2.pdf.

[78] Pablo David Gutierrez et al. "A high performance fingerprint matching system for large databases based on GPU". In: *Information Forensics and Security, IEEE Transactions on* 9.1 (2014), pp. 62–71.

[79] Oluwole Ajala et al. "Fast Fingerprint Recognition Using Circular String Pattern Matching Techniques". In: (2016), pp. 47–52.

[80] Md Aashikur Rahman Azim et al. "A Filter-Based Approach for Approximate Circular Pattern Matching". In: *Bioinformatics Research and Applications*. Springer, 2015, pp. 24–35.

[81] Robert Sedgewick and Kevin Wayne. *Radix Sorts*. 2014. URL: https://www.cs.princeton.edu/~rs/AlgsDS07/18RadixSort.pdf.

[82] James Cryer. *Image analysis and comparison*. 2015. URL: https://huddle.github.io/Resemble.js/.